POLITECNICO DI MILANO
Computer Science and Engineering

Design Document

CodeKataBattle

Software Engineering 2 Project
Academic year 2023 - 2024

07 January 2024
Version 1.0

Authors:
Eliahu Itamar COHEN
Marco GERVATINI
Caterina MOTTI

Professor:
Elisabetta DI NITTO

Contents

Chapter 1

Introduction

1.1 Purpose

The main purpose of the this document is to support the development team in the implementation of the system to be.
It provides an overview of the system architecture used and a breakdown of the various components, which also describes how they interact with each other.
In addition, it is described the implementation, integration, and testing plans.

1.2 Scope

As explained in the RASD, CodeKataBattle is a platform that allows students to improve their development skills in a entertaining way.
The website, that is accessible through any web browser, will provide a dashboard for educators and one for students.
Educators use it to create tournaments, in which students can participate, and battles, in which teams of students can compete against each other.
The platform automatically creates a GitHub repository containing all necessary code. The students must fork the main repository and then they must push their code triggering the CKB platform that automatically updates the battle score of the team. Then the educator can also perform a manual evaluation on each student.
Students can manage their team and see their team score in a battle or personal score in a tournament.
The CKB platform also include gamification badges, a reward that represent the achievements of individual students. Earned badges are visible in the personal profile of the student as well as battles won and tournaments ranking.

1.3 Glossary

1.3.1 Definitions

Term	Definition
Users	Identify both students and educators that are logged in.
Notification	Sent via e-mail, from the system to the recipient.
Code Kata	A set of documents that consist in the description of the project (in which are listed all languages that are acceptable for a solution), the software project, test cases and build automation scripts.

1.3.2 Acronyms

Acronyms	Term
CKB	CodeKataBattle
RASD	Requirements, Analysis and Specification Document of the CKB system
R	Requirement

1.4 Reference documents

- Project assignment specification document.
- Slides of software engineering 2 course on WeBeep.
- CodeKataBattle, Requirements Analysis and Specification Document.

1.5 Document Structure

This document is divided in 5 chapters, as follow:

1. **Introduction**: contains a summary of the CKB system, focusing on the main architectural choices.
2. **Architectural Design**: gives a high level overview of the system, and its partitions in subsystems. Moreover, it describes how these subsystems interacts.
3. **User Interface Design**: explains in detail the functionality from the user's perspective. It analyze more in depth what was presented in the RASD document regarding the user interfaces.
4. **Requirements Traceability**: it draws a parallel between the requirements outlined in the RASD and the components of the system.
5. **Effort Spent**: keeps track of the time spent to realize this document.
6. **References**: keeps track of the software used.

Chapter 2

Architectural Design

2.1 Overview: high-level components and interactions

This section provides an overview of the system's architectural components and their interactions.

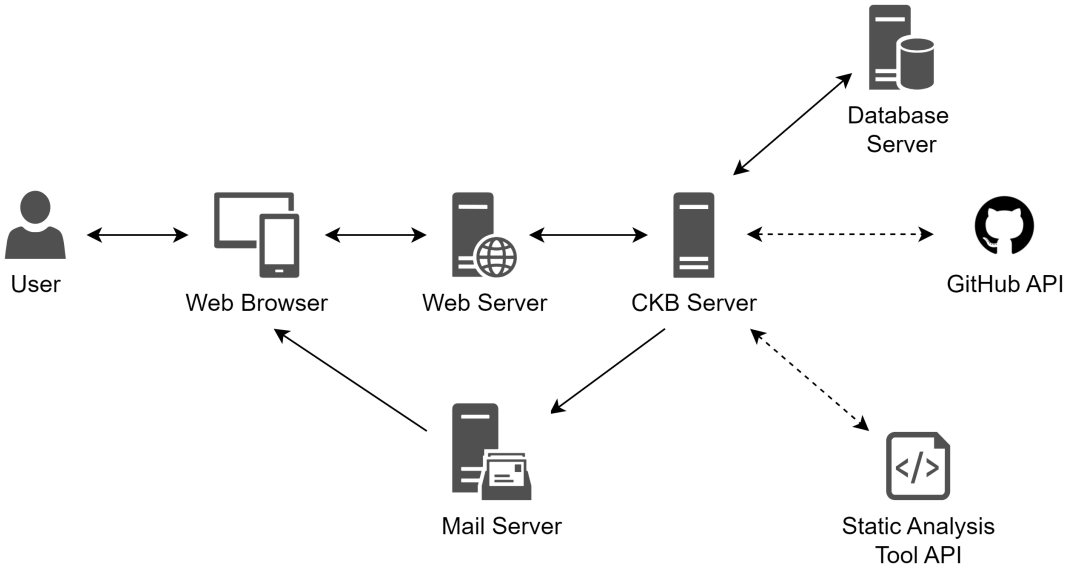

Figure 2.1: CKB system overview

The CKB system will be developed using the client-server paradigm:

- Server side:
  - Web Server: used to communicate with web browsers.
  - CKB Server: where all the logic is located. It communicates with all other server and external tool and API. It is the central point of the system.
  - Database Server: where all information are located.
  - Mail Server: used to send all notifications to the users.
  - GitHub API: used by the system to detect new push in the forked repository of the battles.

  - Static analysis tool: used by the system to retrieve the quality level of the student's sources, in order to perform the automatic evaluation.

- Client side:

  - Web browser: used by all users in order to enter the CKB website.

The application will be developed on a three-tiered architecture where the layers (presentation, application, data) are divided into three different tiers.
The client tier is responsible only of the presentation layer, therefore a thin-client has been adopted since the required client-side functionality are limited. The application tier is responsible of the application layer, it receives the request from the clients and handles them. It communicates with the data tier that is responsible of the data layer, it is able to access the data in the database.
Further details on the system components and their interactions will be explained in detail in the following sections.

2.2 Component view

In the following section it is show the component view of the entire system, as well as all internal and external interactions.
In figure 2.2 is represented all the component and the external interactions. Furthermore, in figure 2.3 is represented in detail the internal structure of the CKB server, which contains the business logic of the system.

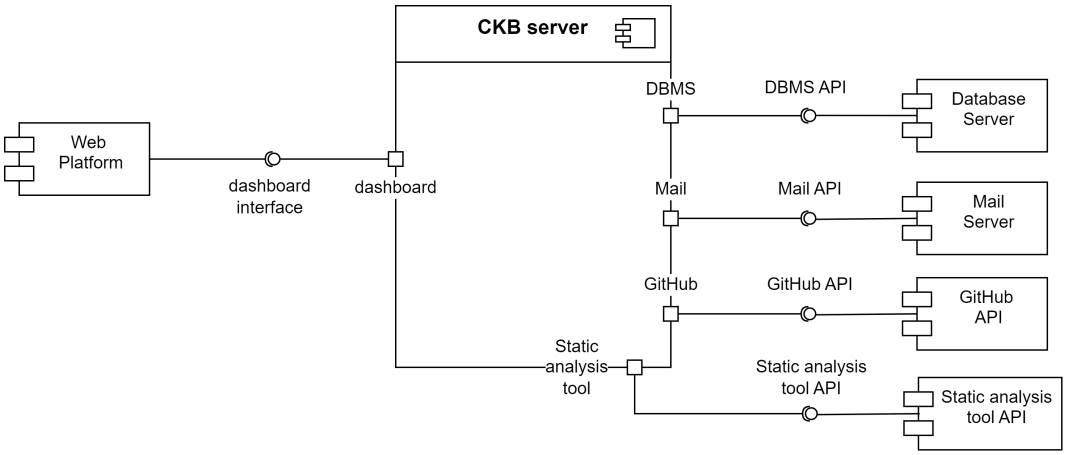

Figure 2.2: Component diagram

The external components of the CKB system represented in figure 2.2 are:

- Web platform: is the presentation layer of the website that allow all users to use the CKB functionalities.
- Mail service: used by the CKB system to send notification to all registered users when needed.
- Database: where all the data are stored. It communicate with the DBMS running on the database.
- GitHub API: it trigger the CKB system when there is a new push in any forked repository of a battle, in order to perform the automatic evaluation.
- Static analysis tool API: it is used by the CKB system in order to retrieve the quality level of the student's sources, in order to perform the automatic evaluation.

The internal components of the CKB server represented in figure 2.3 are:

- Dashboard manager: handles all the interfaces offered in the web platform and it is in charge of providing the right interface to each user.
- Tournament manager: handles the main features that allows educators to create and manage a tournament. It also manages all the tournament rankings.

- Battle manager: handles the main features that allows educators to create and manage a battle, and students to manage their teams. It also manages all the battle real-time rankings.
- User access manager: handles the log in and sign up functions. It also manages all students profile.
- Link manager: manages all interactions that pass through a link received via email, such as subscribe in a tournament, enroll in a battle, join a tournament or a team.
- Model: almost all components interacts with the model, since it is the entry point to the data stored in the database.
- Entity manager: it deals with all the data management needed by the system.
- Mail manager: interfaces with the Mail API to send notifications via email to all users.
- Automatic evaluation manager: it deals with both the GitHub API and the static analysis tool API, in order to modify the real-time rank of a battle.

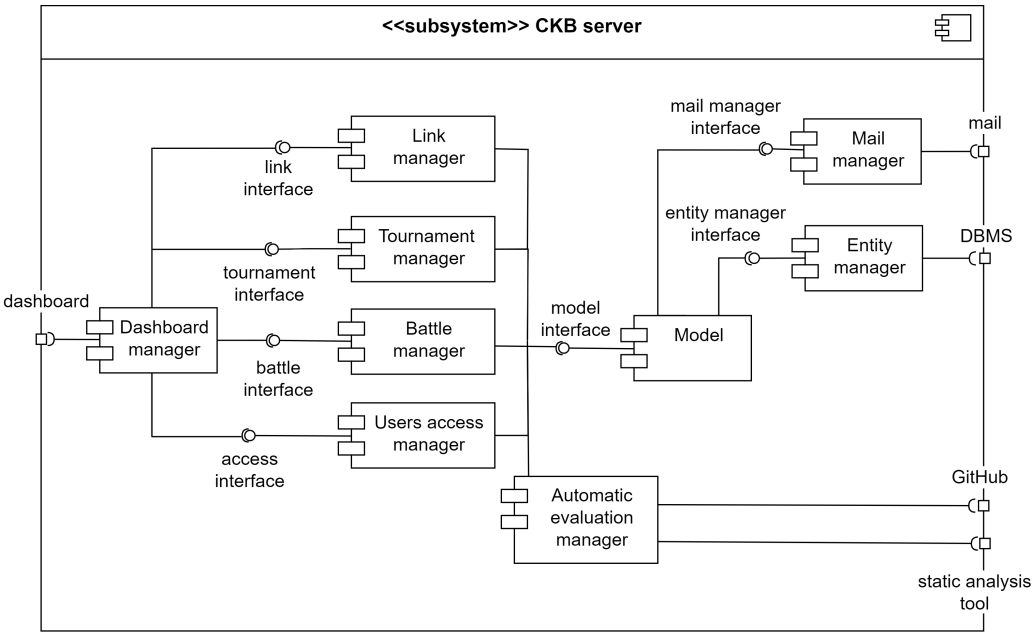

Figure 2.3: Component CKB server diagram

A further explanation of the relevant internal components follows.

Dashboard manager

In figure 2.4 is represented the dashboard manager with its sub-components. The input manager handles all the different requests that can be made through the web platform. The student and the educator action manager manages respectively, the student and educator requests.

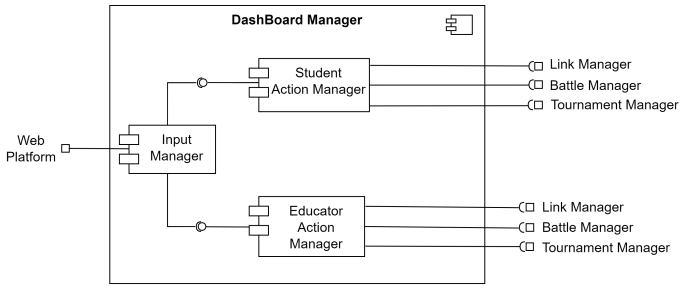

Figure 2.4: Dashboard manager

User access manager

In figure 2.5 is represented the user access manager with its sub-components. The input manager handles all the different requests that can be made through the web platform. Then two different sub-components are responsible for the sign up and the log in functions.

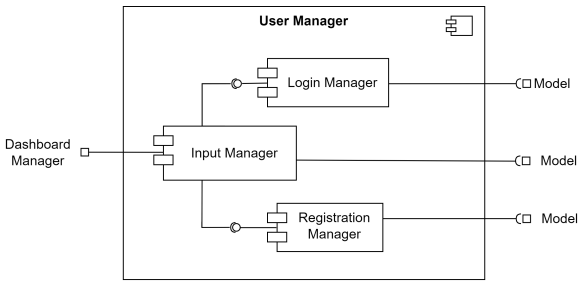

Figure 2.5: User access manager

Tournament manager

In figure 2.6 is represented the tournament manager with its sub-components. It manages all functions that involve a tournament, from both the student and the educator point of view.

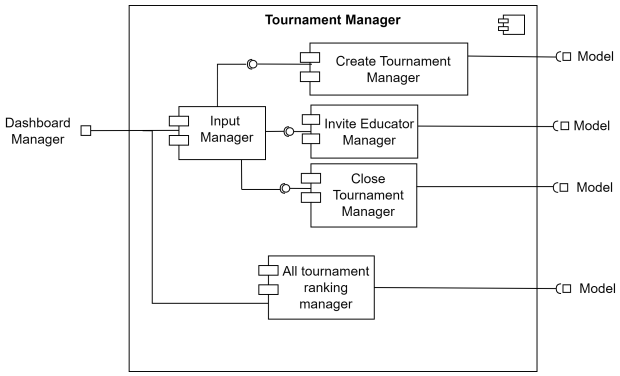

Figure 2.6: Tournament manager

Battle manager

In figure 2.7 is represented the battle manager with its sub-components. It manages all functions that involve a battle, from both the student and the educator point of view.

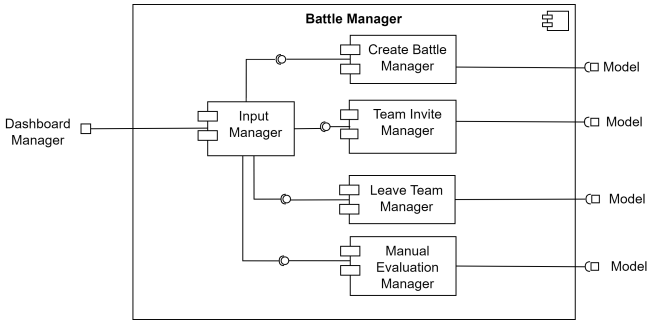

Figure 2.7: Battle manager

Link manager

In figure 2.8 is represented the link manager with its sub-components. It manages all functions that involve a link received by email, from both the student and the educator point of view.

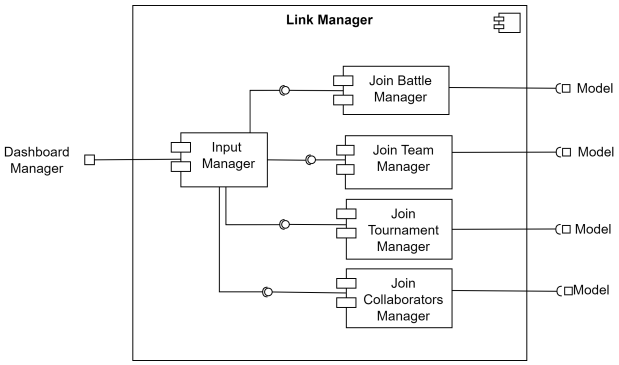

Figure 2.8: Link manager

Automatic evaluation manager

In figure 2.9 is represented the automatic evaluation manager with its sub-components. It manages all functions that involve the GitHub API and the static analysis tool, in order to perform the automatic evaluation.

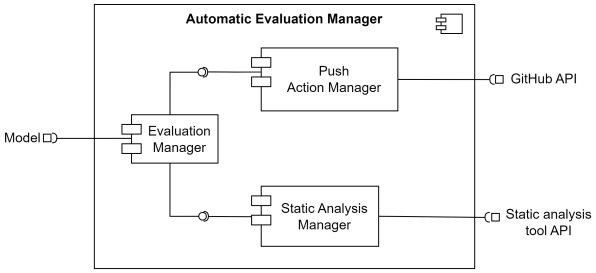

Figure 2.9: Automatic evaluation manager

2.3 Deployment view

As previously stated in section 2.1 the CKB system will be implemented through a three-tier architecture as shown in figure 2.10.
The three-tired architecture is a well-established software application architecture that organizes the system into three logical and physical computing layers:

- **Presentation layer/tier**: it is the user interface where the user interacts with the application. It communicates with the application layer in order to retrieve all necessary information requested by the user. Its main purpose is to display information and collect information from the user.
- **Application layer/tier**: it manages all system functionalities. All information collected in the application layer are processed using a specific business logic. It communicates with the data layer in order to store, delete or modify useful data.
- **Data layer/tier**: it includes the database of the system and all necessary mechanism to extract and evaluate data.

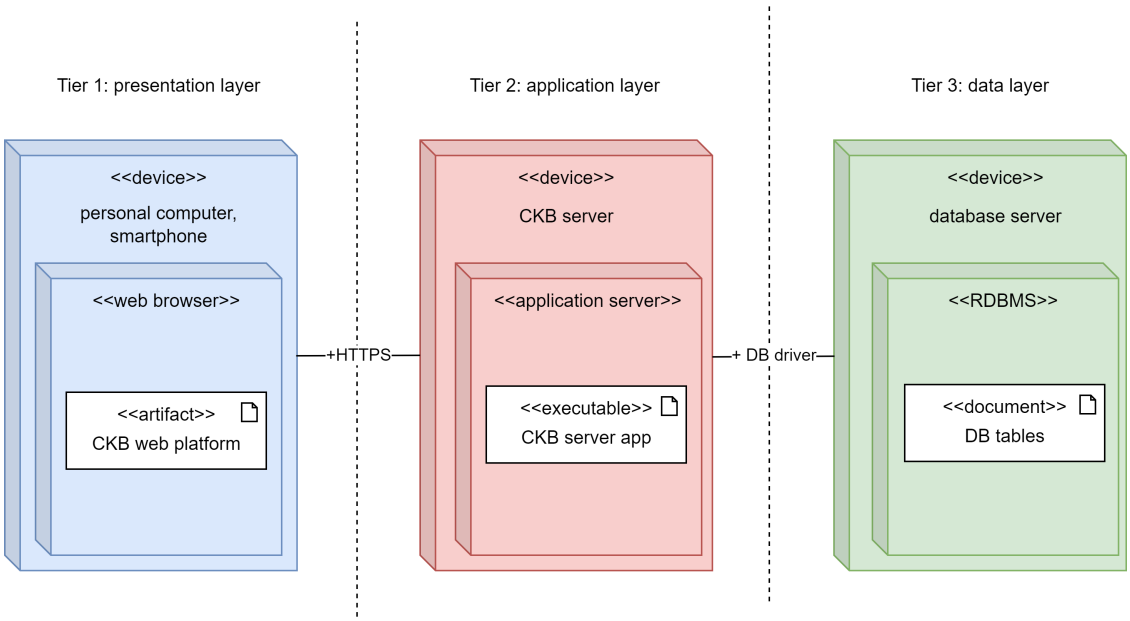

Figure 2.10: Deployment diagram

2.4 Component interface

This section lists all the methods that each component interface provides to the other components.

- **Dashboard Interface**:
  - signupRequest(name, surname, username, password, email, role)
  - loginRequest(username, password)
  - createTournament(title, subDeadline, badges)
  - inviteCollaborator(username, tournamentId)
  - showOngoingTournaments()
  - showRanking(tournamentId)
  - enterTournament(username, tournamentId)
  - closeTournament(tournamentId)
  - createBattle(name, codeKata, subDeadLine, finalDeadline, minTeam, MaxTeam, tournamentId)
  - leaveTeam(username, teamId)
  - inviteStudent(teamId, username)
  - showRanking(battleId)
  - subscribeTournamentEducator(tournamentId, username)
  - subscribeTournament(tournamentId, username)
  - subscribeBattle(battleId, username)
  - subscribeTeam(teamId, username)
  - createBadge(title, description, rules)
  - getUser(username)
  - manualEvaluation(username, battleId, score)
  - showOngoingBattles()
  - confirmation(username)
- **Tournament Interface**:
  - createTournament(title, subDeadline, badges)
  - inviteCollaborator(username, tournamentId)
  - showOngoingTournaments()
  - showRanking(tournamentId)
  - closeTournament(tournamentId)
  - createBadge(title, description, rules)
- **Battle Interface**:

  - createBattle(name, codeKata, subDeadLine, finalDeadline, minTeam, MaxTeam, tournamentId)
  - leaveTeam(username, teamId)
  - inviteStudent(teamId, username)
  - showOngoingBattles()
  - showRanking(battleId)
  - manualEvaluation(username, battleId, score)

- **Link Interface**:
  - subscribeTournamentEducator(tournamentId, username)
  - subscribeTeam(teamId, username)
  - subscribeBattle(battleId, username)
  - subscribeTournament(tournamentId, username)

- **User Access Interface**:
  - signupRequest(name, surname, username, password, email, role)
  - loginRequest(username, password)
  - getUser(username)
  - confirmation(username)

- **Model Interface**:
  - signupRequest(name, surname, username, password, email, role)
  - createUser(name, surname, username, password,email, role)
  - loginRequest(username, password)
  - createTournament(title, subDeadline, badges)
  - inviteCollaborator(username, tournamentId)
  - showOngoingTournaments()
  - showRanking(tournamentId)
  - closeTournament(tournamentId)
  - createBattle(name,codeKata, subDeadLine, finalDeadline, minTeam, MaxTeam, TournamentId)
  - leaveTeam(username,teamId)
  - inviteStudent(teamId,username)
  - showRanking(battleId)
  - subscribeTournamentEducator(tournamentId, username)
  - subscribeTournament(tournamentId, username)
  - subscribeBattle(battleId, username)
  - subscribeTeam(teamId, username)

  - getParameters(battleId)
  - evaluate(code, battleId)
  - eval_tests()
  - eval_time()
  - createBadge(title, description, rules)
  - getUser(username)
  - manualEvaluation(username, battleId, score)
  - confirmation(username)
  - showOngoingBattles()
  - assign_badges()
  - generate_final_ranking()

- **Entity Manager Interface**:
  - checkUsername(username)
  - checkUserInfo(username, password)
  - createUser(name, surname,username,password,email,role)
  - createTournament(title, subDeadline, badges)
  - createBattle(name, codeKata, subDeadLine, finalDeadline, minTeam, MaxTeam, tournamentId)
  - leaveTeam(username, teamId)
  - getRanking(tournamentId)
  - getTournaments()
  - getBattles(date)
  - closeTournament(tournamentId)
  - newCollaborator(tournamentId, username)
  - newPartecipant(battleId, username)
  - newTeamMember(teamId,username)
  - updateRankings(tournamentId, battleId)
  - createBadge(title, description, rules
  - manualEvaluation(username, battleId, score)

- **Mail Manager Interface**:
  - sendSignUpMail(username, email)
  - sendCollaboratorMail(tournamentId, mail)
  - sendTeamInvitationMail(teamId, mail)
  - sendFinalRankingMail(tournamentId, all)
  - sendNewBattleMail(battleId, all)
  - sendNewTournamentMail(tournamentId, all)

2.5 Run-time view

Sign up

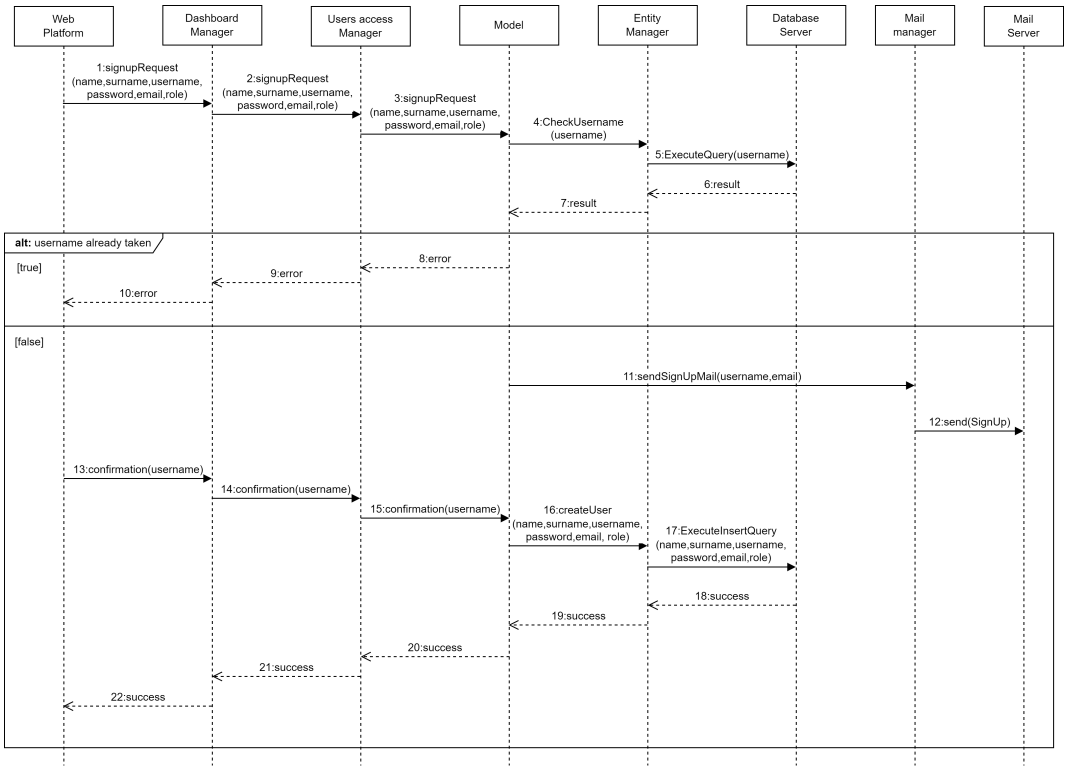

Figure 2.11: Sign up runtime diagram

Log in

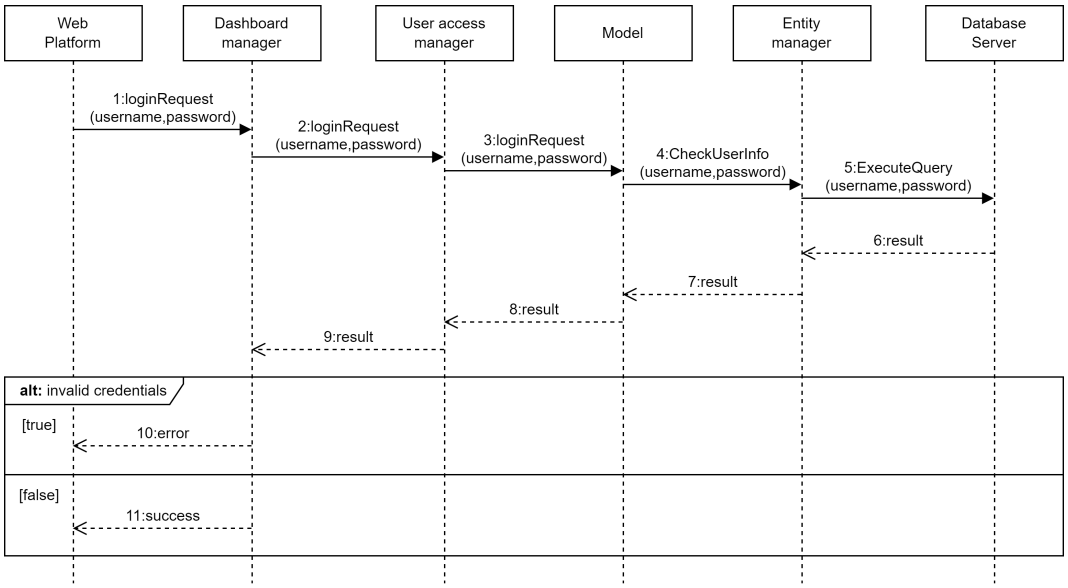

Figure 2.12: Log in runtime diagram

Create tournament

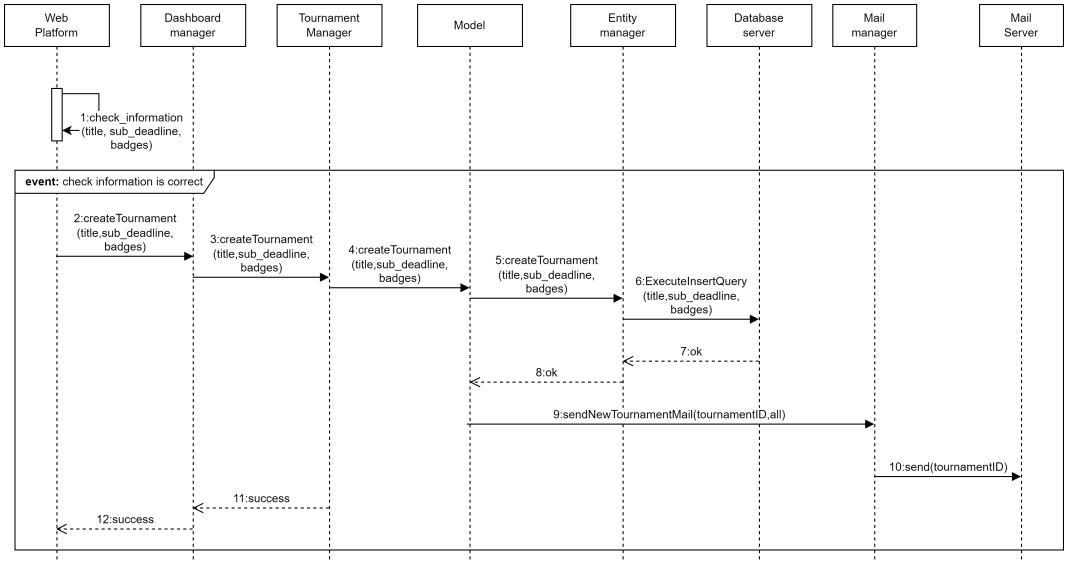

Figure 2.13: Create tournament runtime diagram

Invite educator

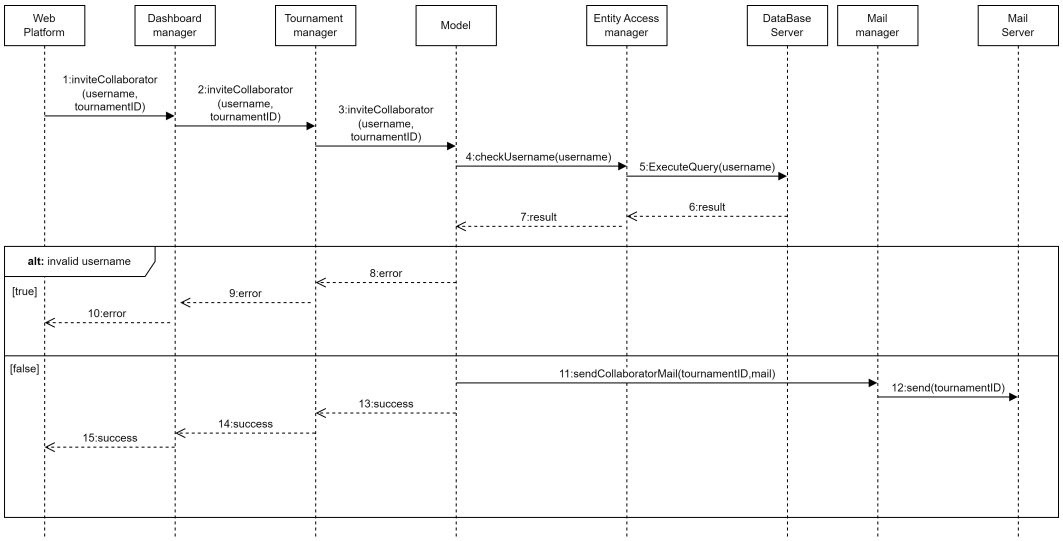

Figure 2.14: Invite educator runtime diagram

Educator accepts invite

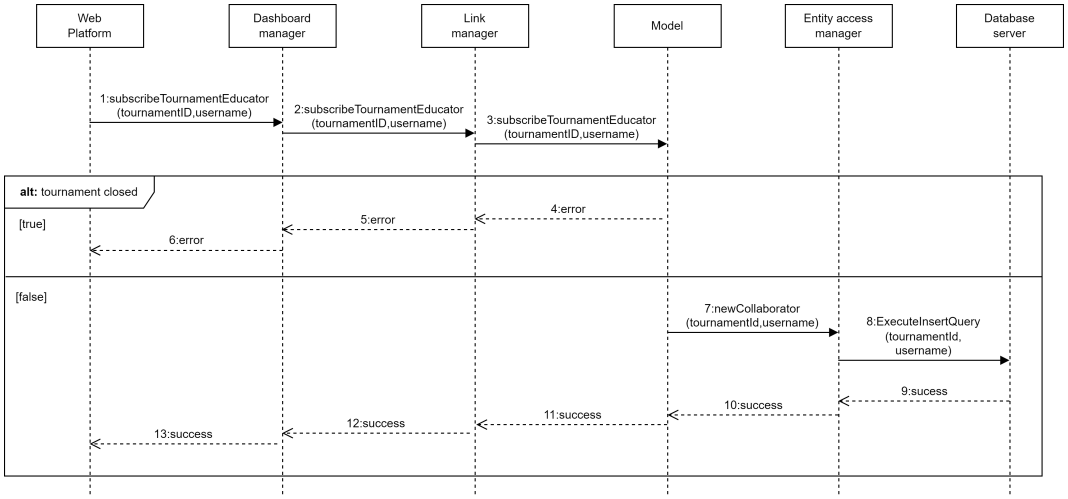

Figure 2.15: Educator accepts invite runtime diagram

Create badge

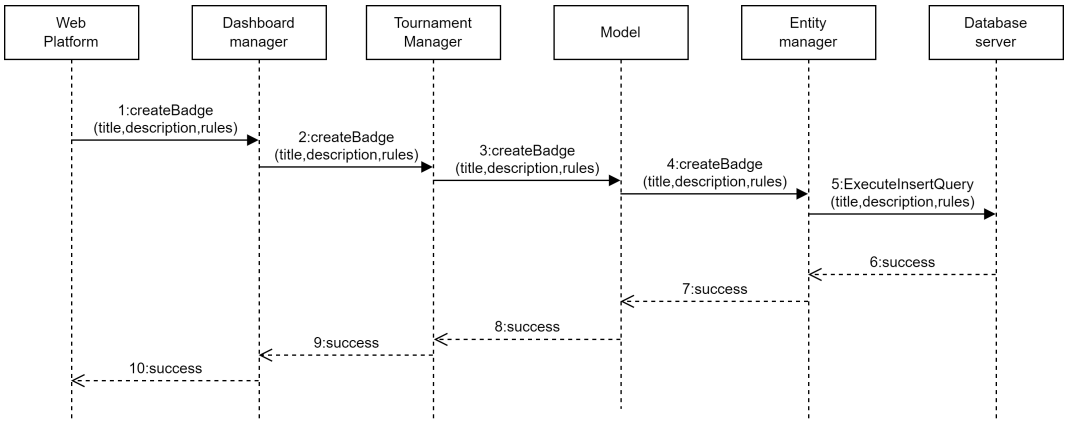

Figure 2.16: Create badge runtime diagram

Close tournament

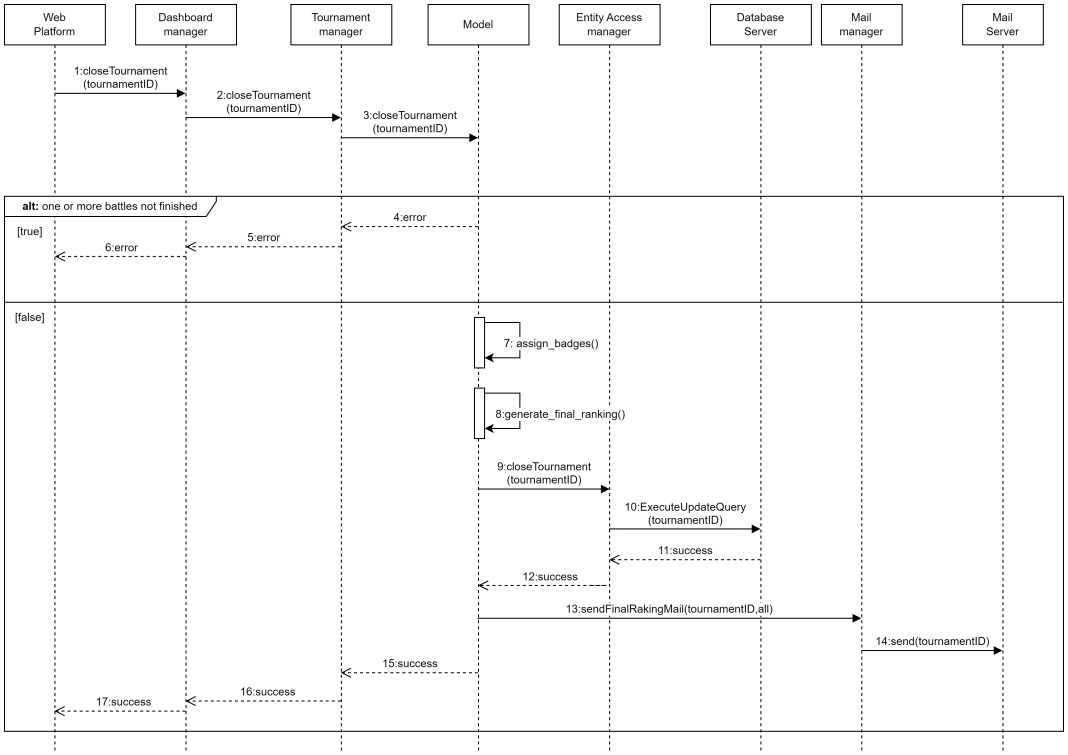

Figure 2.17: Close tournament runtime diagram

Create battle

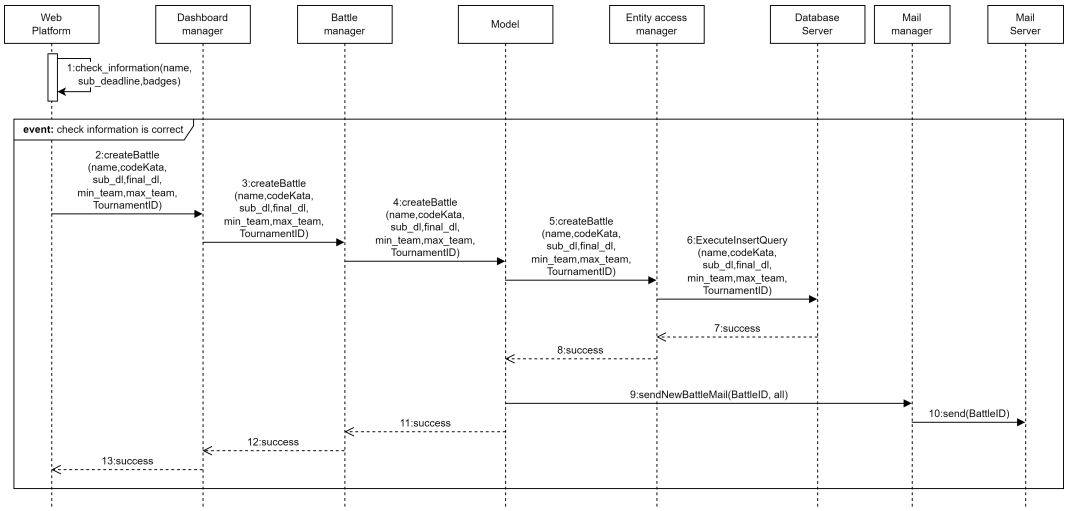

Figure 2.18: Create battle runtime diagram

Subscribe in a tournament

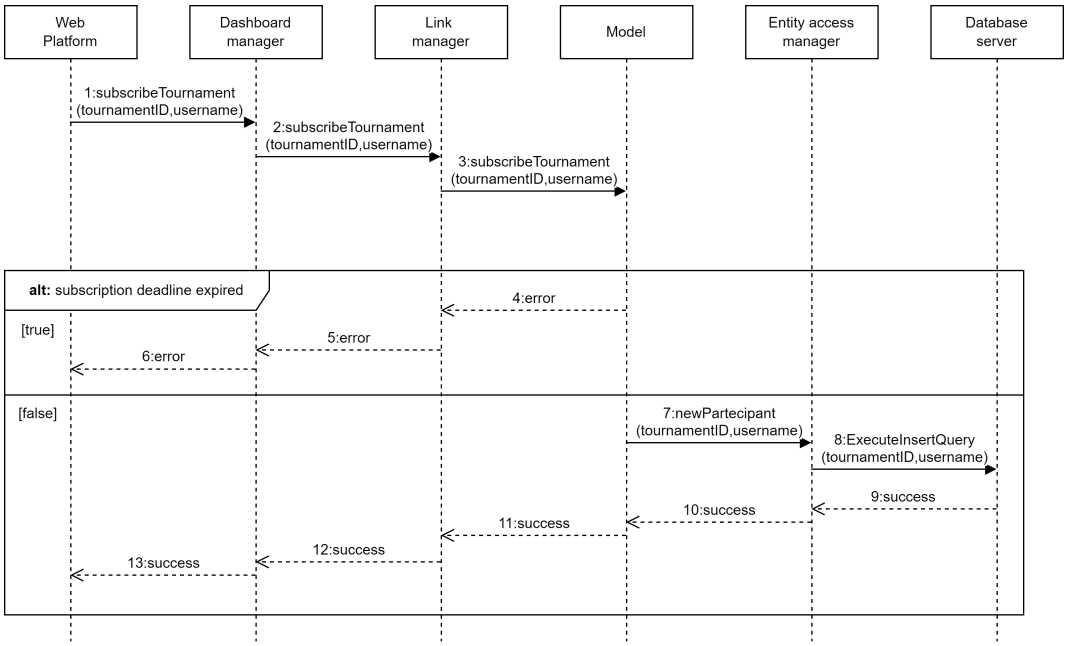

Figure 2.19: Subscribe in a tournament runtime diagram

Enroll in a battle

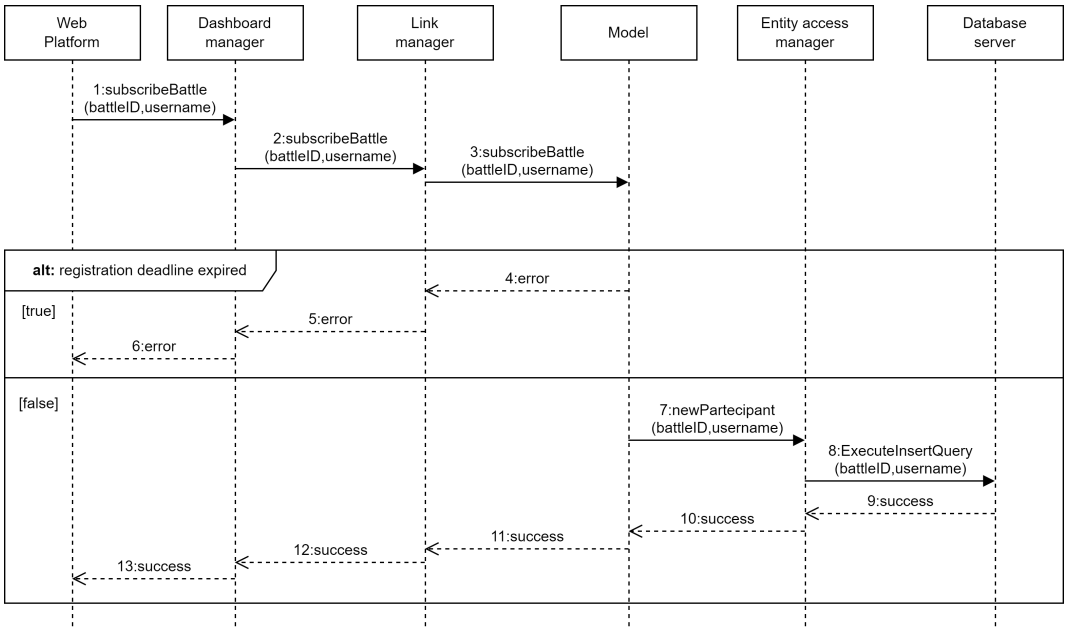

Figure 2.20: Enroll in a battle runtime diagram

Invite student in a team

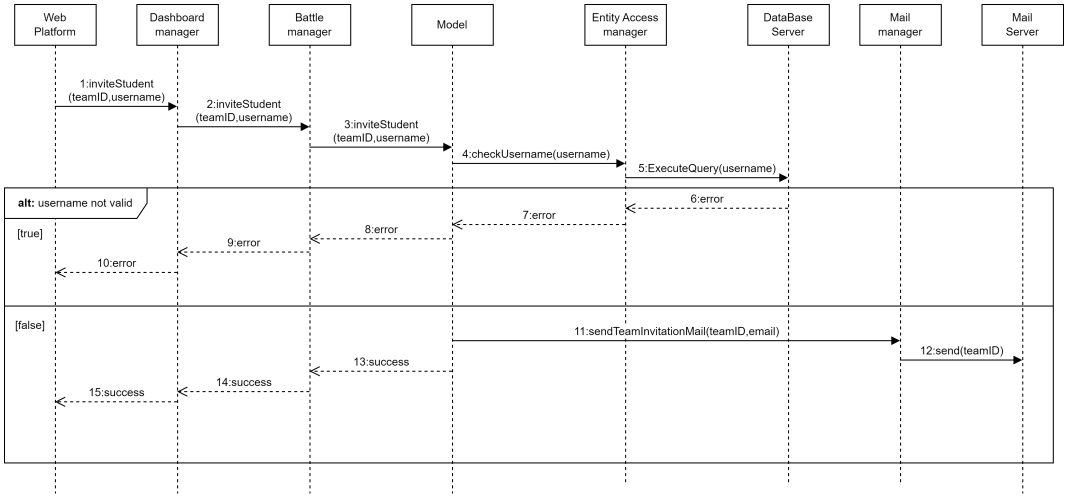

Figure 2.21: Invite student in a team runtime diagram

Student accepts invite

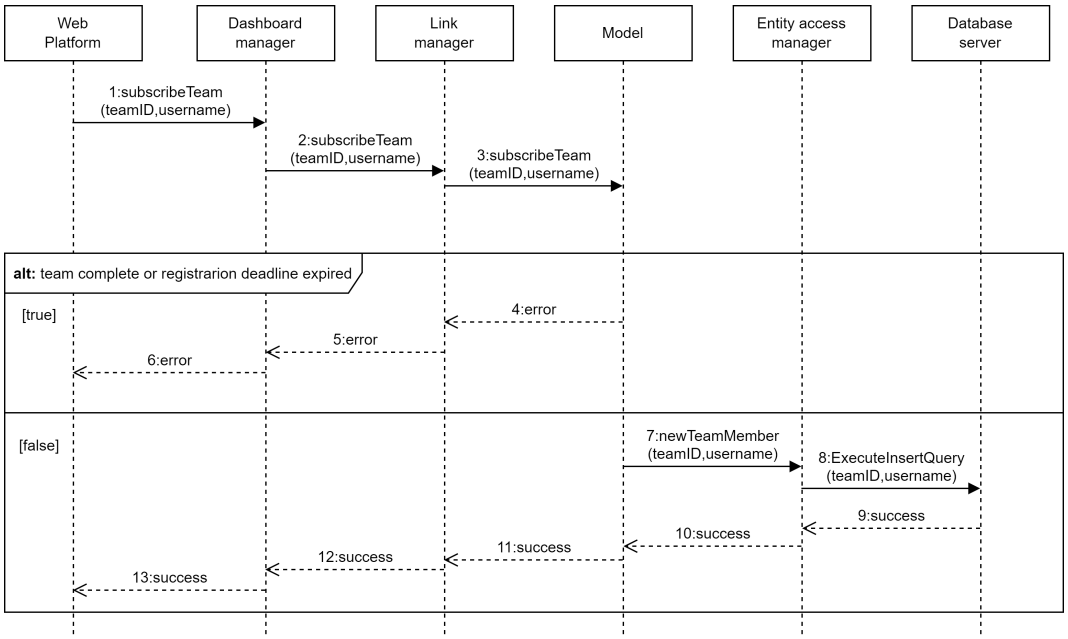

Figure 2.22: Student accepts invite runtime diagram

Student leaves team

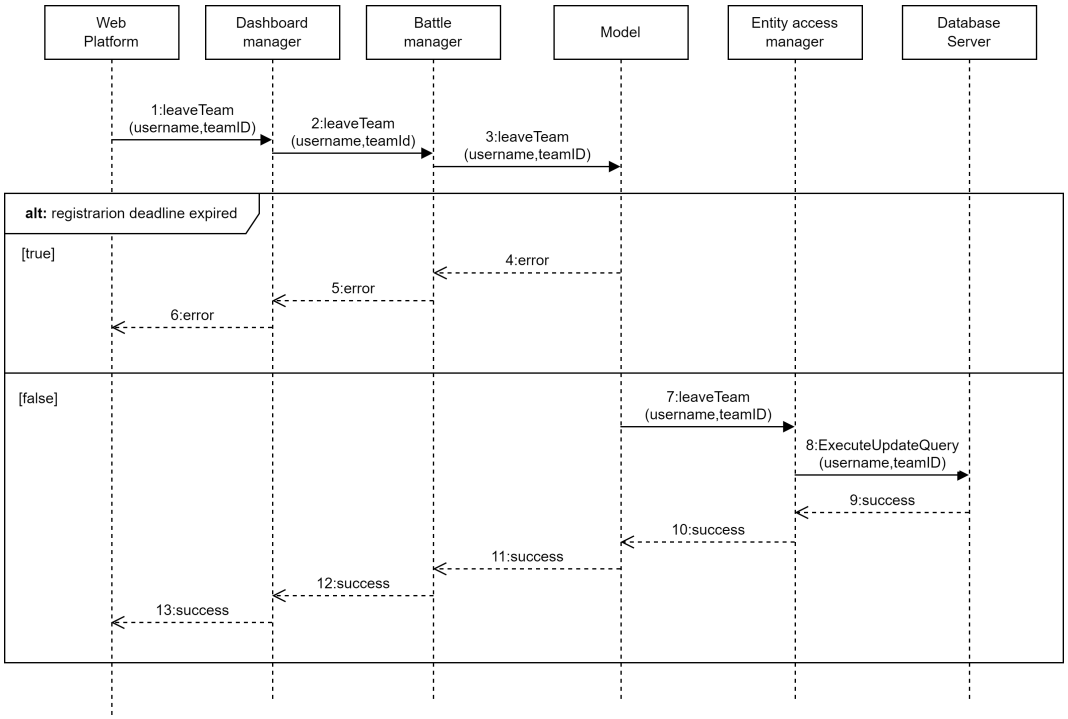

Figure 2.23: Student leaves invite runtime diagram

Perform manual evaluation

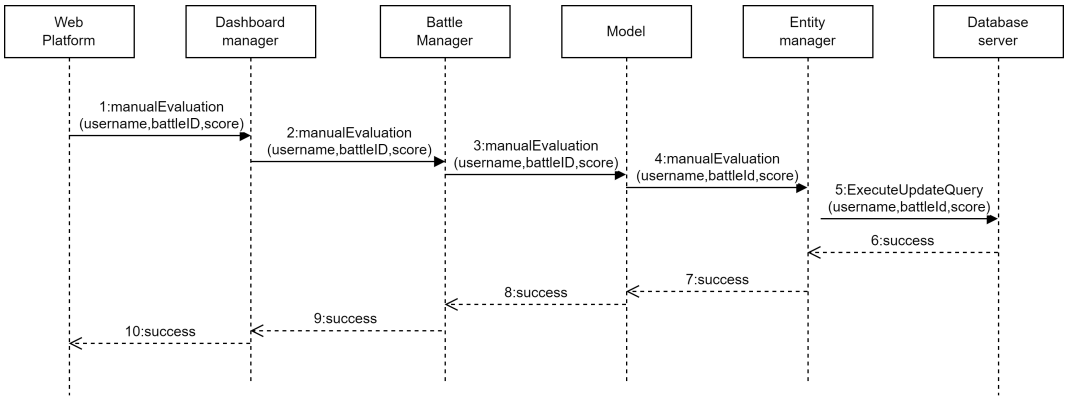

Figure 2.24: Perform manual evaluation runtime diagram

Perform automatic evaluation

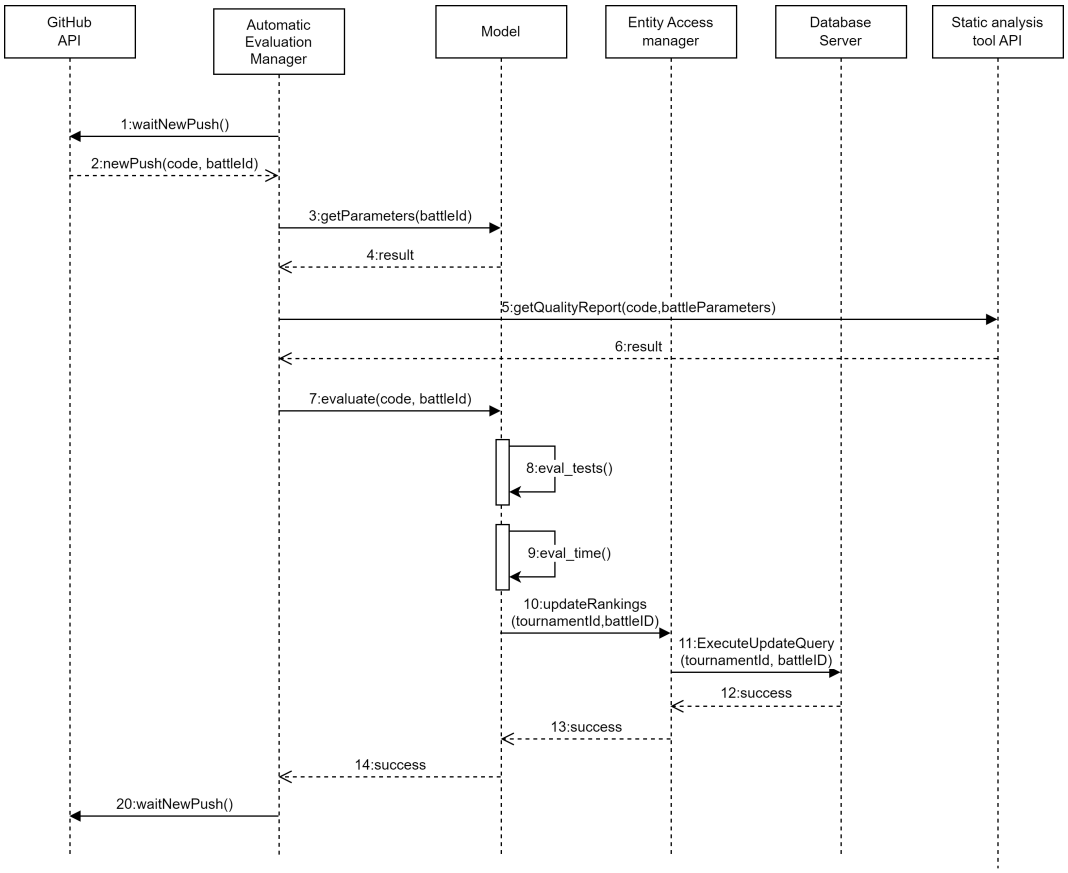

Figure 2.25: Perform automatic evaluation runtime diagram

Visualize battle ranking

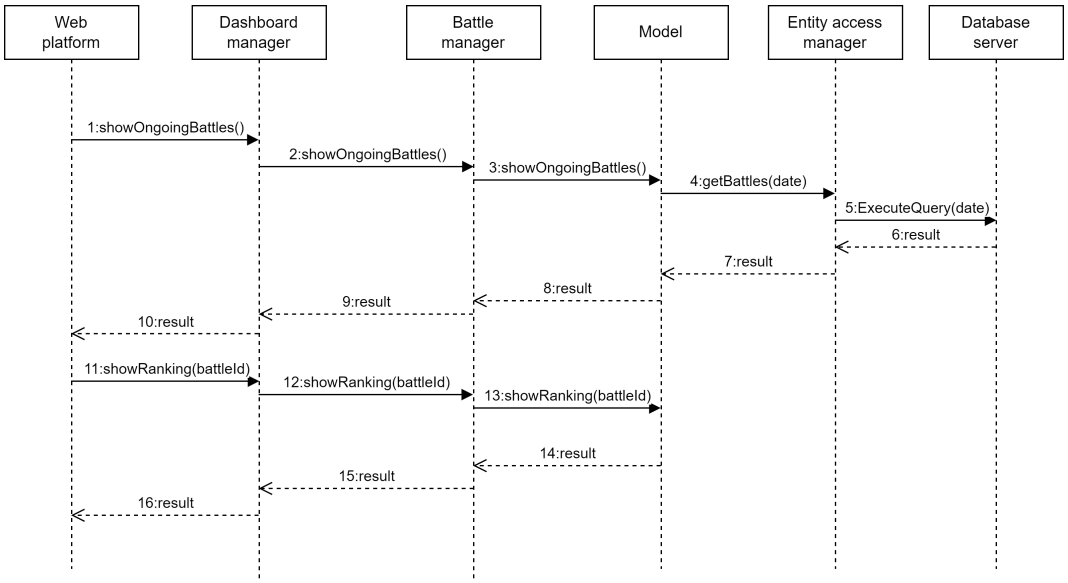

Figure 2.26: Visualize battle rankgin runtime diagram

Visualize all ongoing tournament with ranking

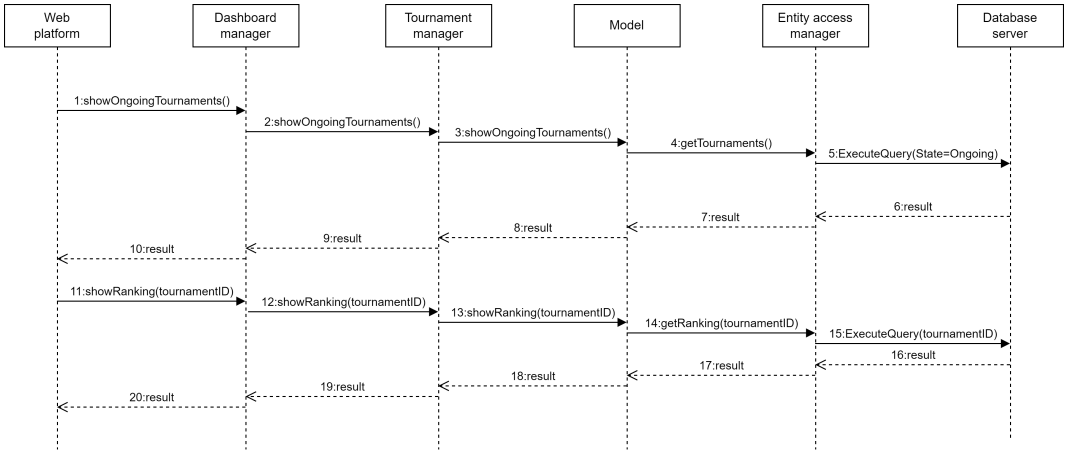

Figure 2.27: Visualize all ongoing tournament with ranking runtime diagram

See a student's profile

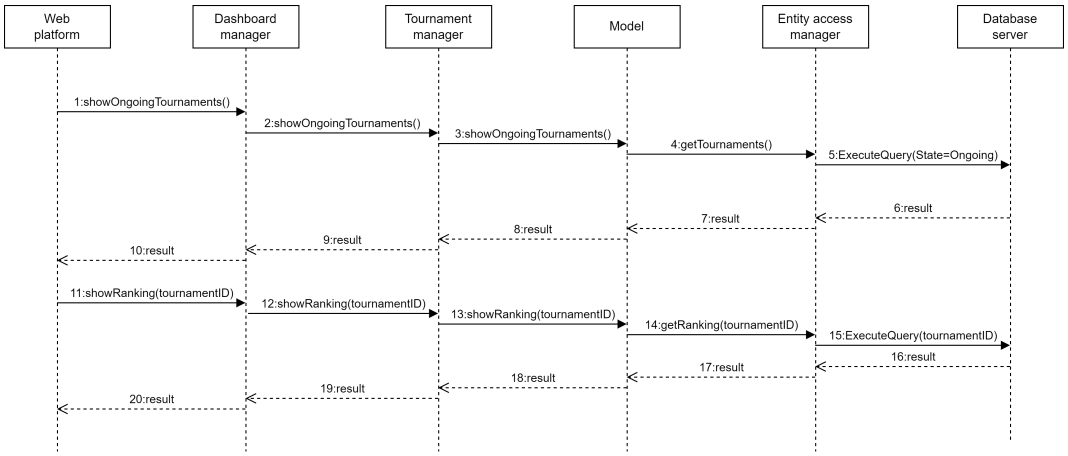

Figure 2.28: See a student's profile runtime diagram

2.6 Selected architectural styles and patterns

Client-Server Architecture

As previously mentioned in section 2.1 the CKB system will be developed based on the client-server architecture. Client and server are two separate objects that communicate over a network: a client makes a request for a service, the server receives this request, process it, and sends back a response.

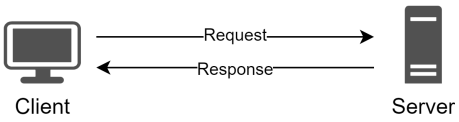

Figure 2.29: Client-Server paradigm

3-tier Architecture

As previously mentioned in section 2.1 and in section 2.3 the CKB system will adopt the three-tier architecture, and for this reason it will be developed in three independent layers (or tiers).

Model View Controller Pattern

The software will be implemented based on a MVC patter, that divides the program logic into three interconnected parts:

- Model: it is the core element of the entire system that directly manage the logic of the CKB platform.
- View: it manages all user interfaces.
- Controller: it handles all inputs from the view and manipulates the objects in the model.

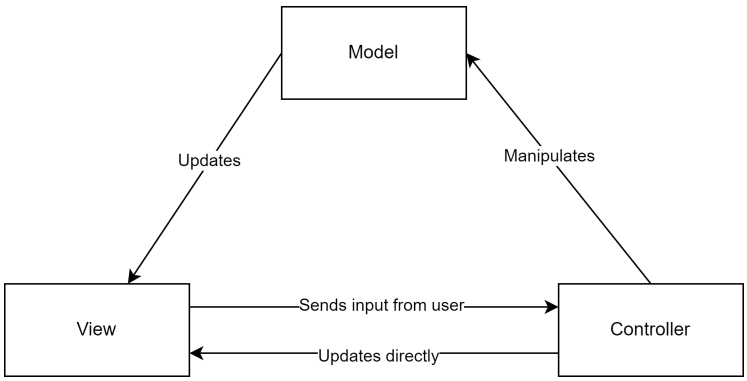

Figure 2.30: Model-View-Controller paradigm

Facade pattern

The dashboard component is implemented using the facade pattern, which hide the system's complexity from the client and offers a more user-friendly interface. In this way the client communicates only with the dashboard interface, that manages all inputs. The dashboard manager communicates with the appropriate sub-component based on the client request.

2.7 Other design decision

Relational DBMS

The data layer, as mentioned in section 2.3, consists of a relational DBMS. A MySQL DBMS will be used since it is an open-source relational database management system (RDBMS).
In figure 2.31 shows the database structure.
For clarity, the "password" attribute in the user table is represented, but will be encrypted.

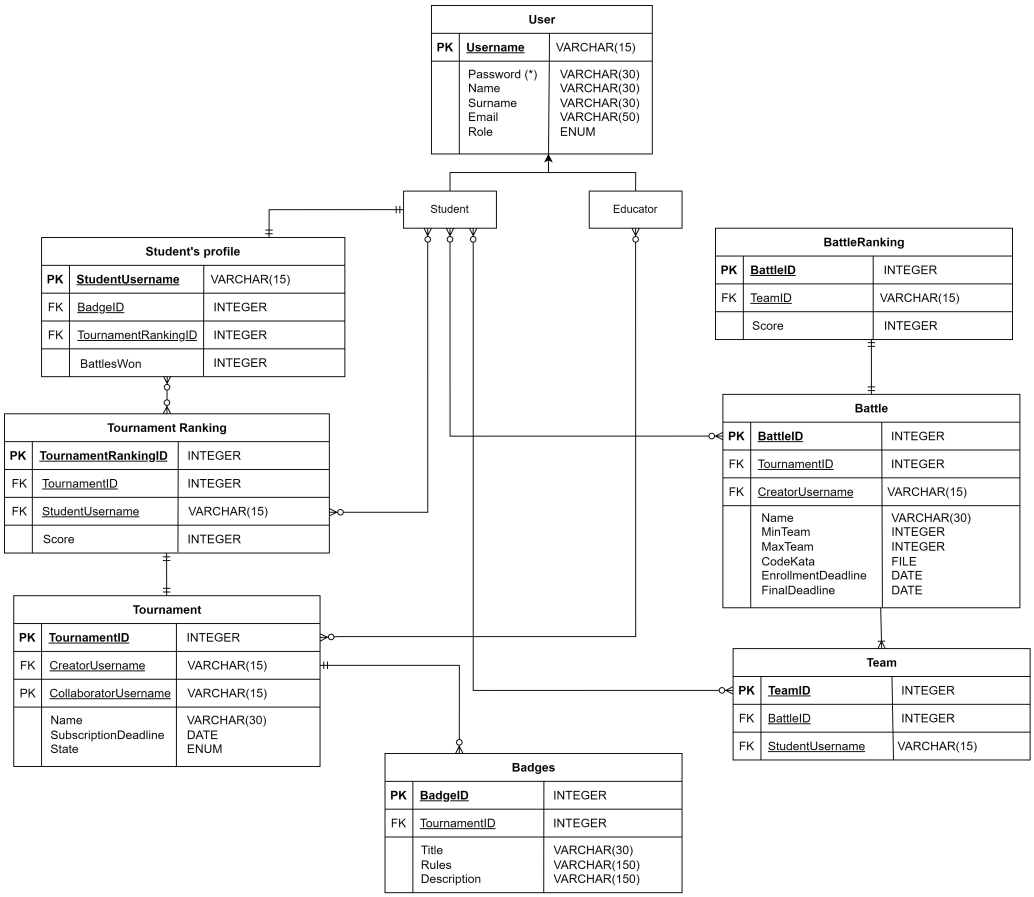

Figure 2.31: Database structure

Chapter 3

User Interface Design

3.1 Mockups

The only way to use the CKB platform is through the website. As previously mentioned in the RASD document the interfaces will be different according to the user who is using them, as the provided functionality are different.

Used by all users.

In figure 3.1 is represented the initial page that all users visualize every time they open the website. The user can subsequently choose between "log in" in figure 3.2, if he is already registered, or "sign up in figure 3.3.

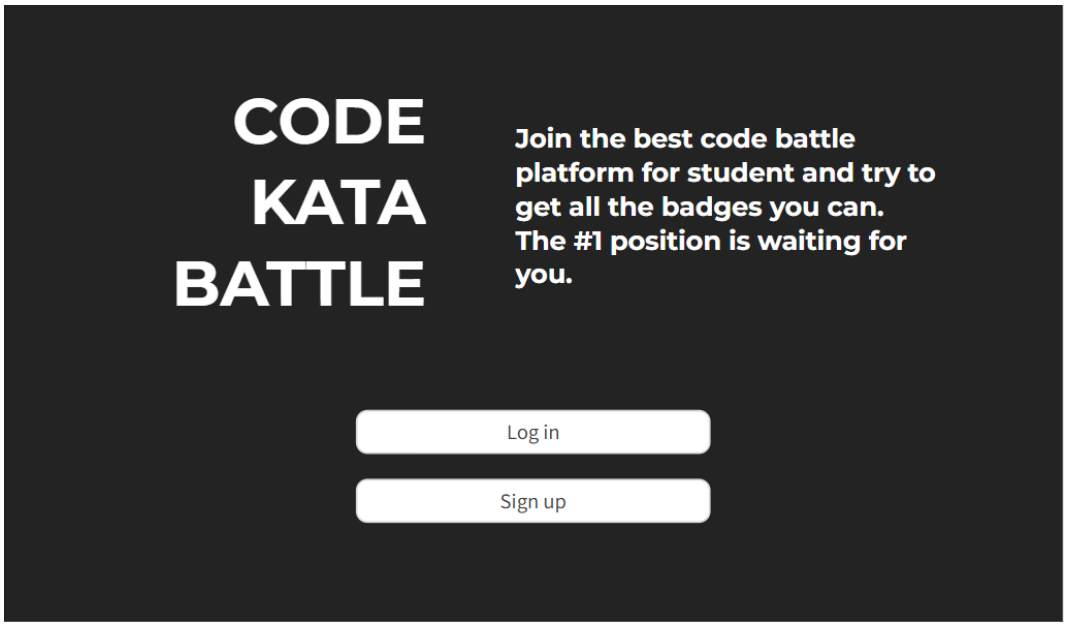

Figure 3.1: Front Page

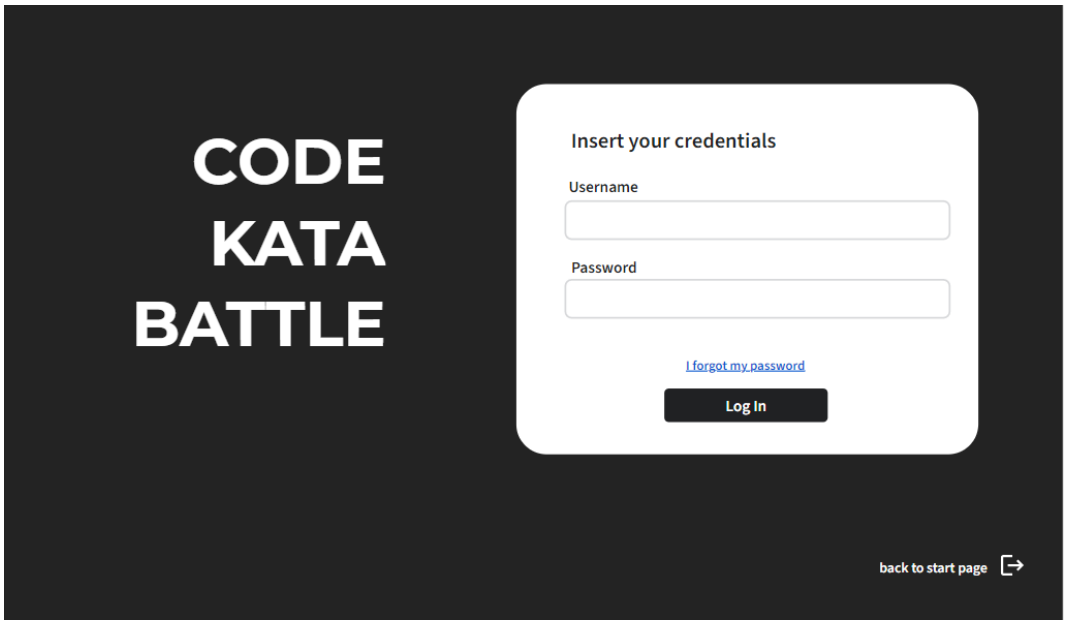

Figure 3.2: Log in

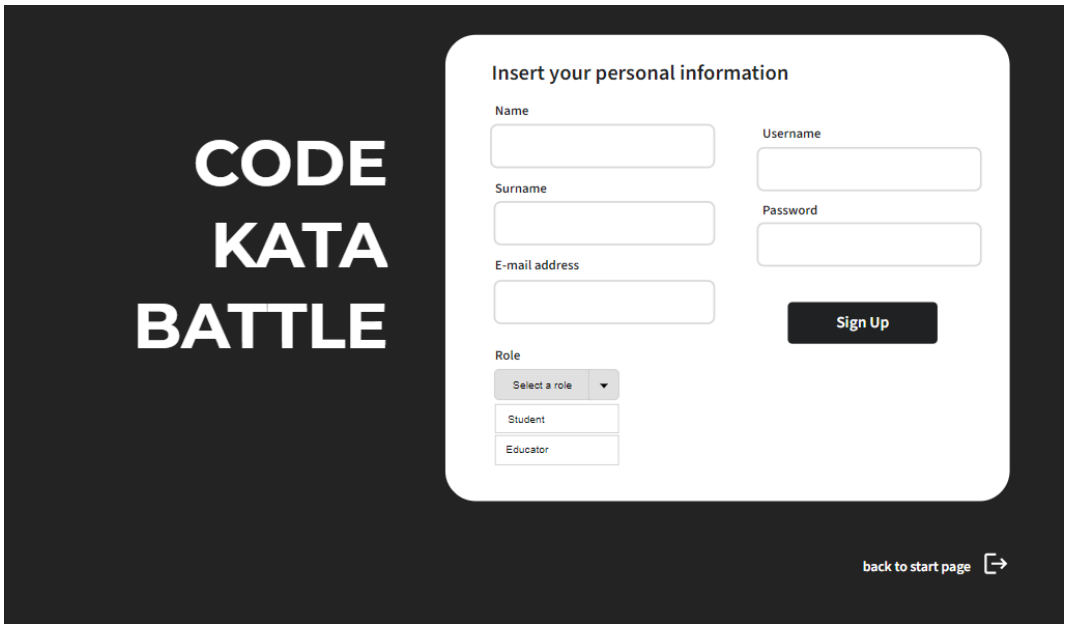

Figure 3.3: Sign up

Another feature available to both students and educators is the ability to view all tournaments and their rankings. By selecting a tournament in the all tournaments page in figure 3.4, the user is able to see its ranking and all active badges as in figure 3.5.

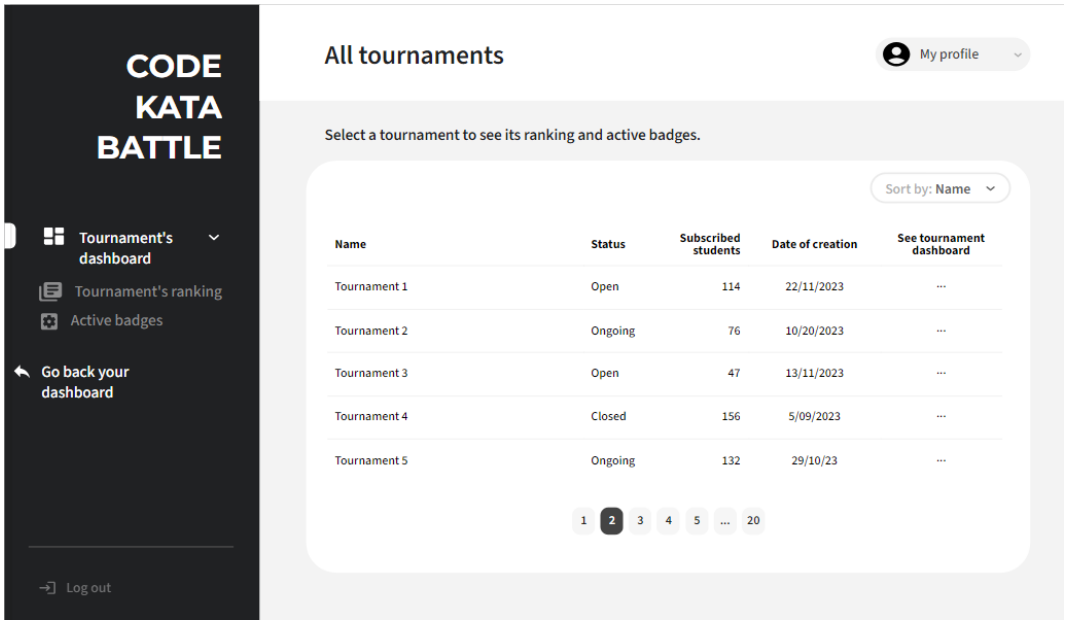

Figure 3.4: All tournaments

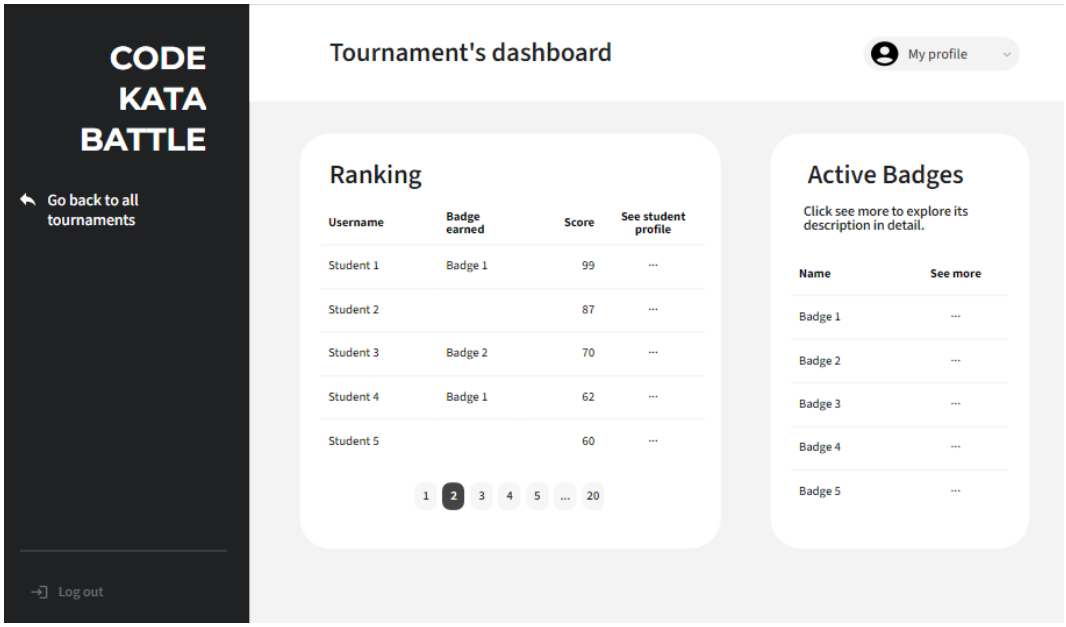

Figure 3.5: Tournament dashboard

To view a student's profile, the users will have a function in the general dashboard. They must type the username in the field and then search for it. In the personal

profile of a student, each user can view his/hers tournament ranking, battles won and all the badges collected.

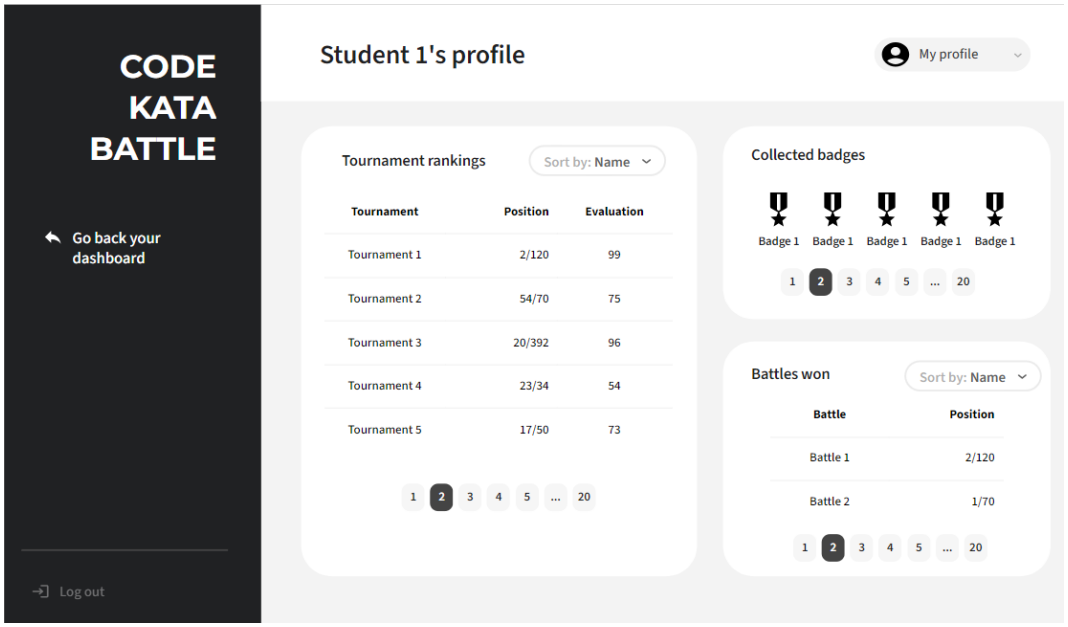

Figure 3.6: Student profile

Used by the educators.

From the general dashboard the educator will be able to use the CKB platform in all its functionality.

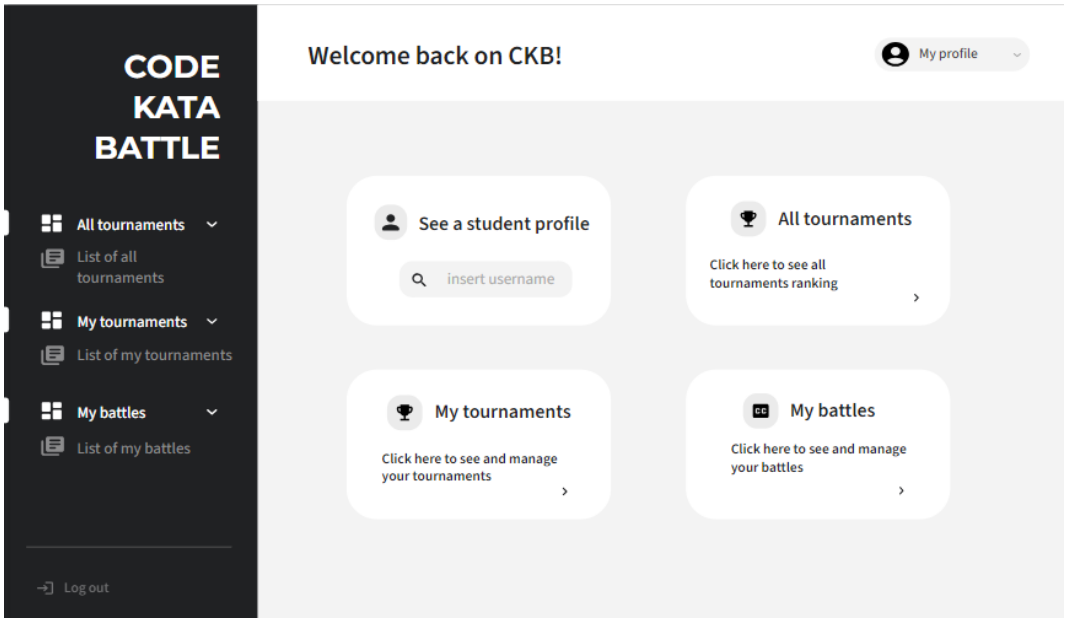

Figure 3.7: Educator's dashboard

By compiling all the fields in the "Create a tournament" page in figure 3.8 the educator is able to create a tournament and badges.

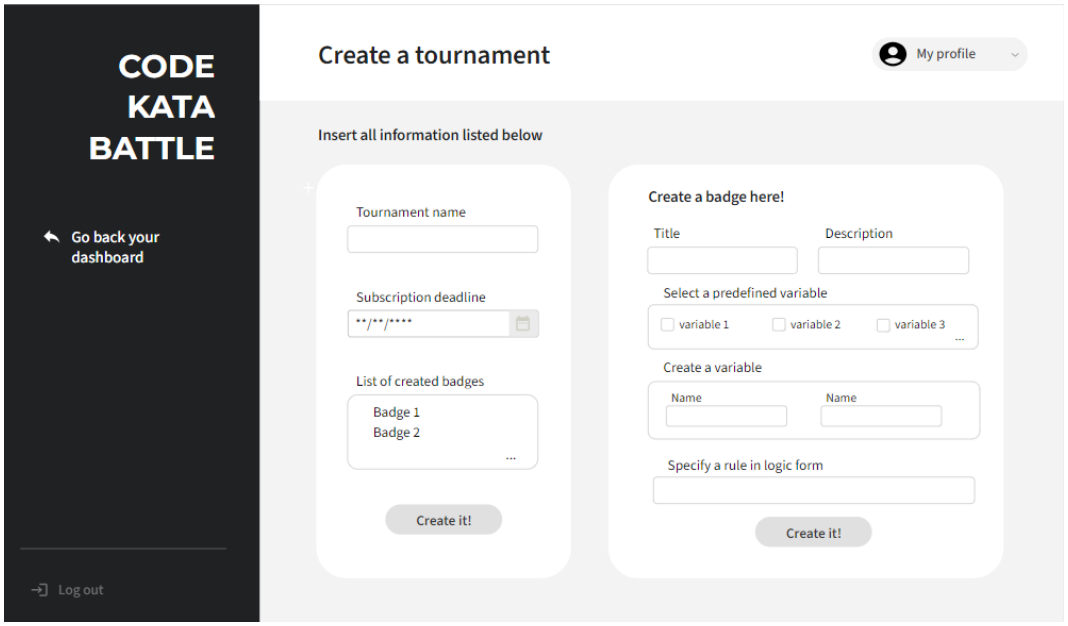

Figure 3.8: Create tournament

From the "My tournaments" page in figure 3.9 the educator is able to manage them. By clicking on one tournament the educator will enter the page "manage tournament" in figure 3.10.

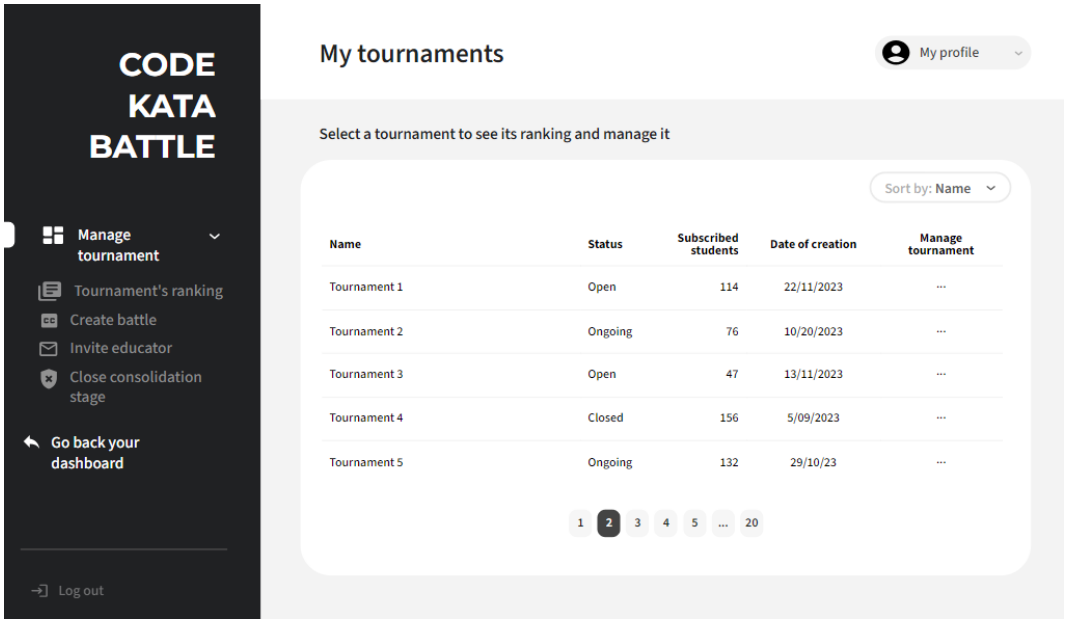

Figure 3.9: Educator's tournaments

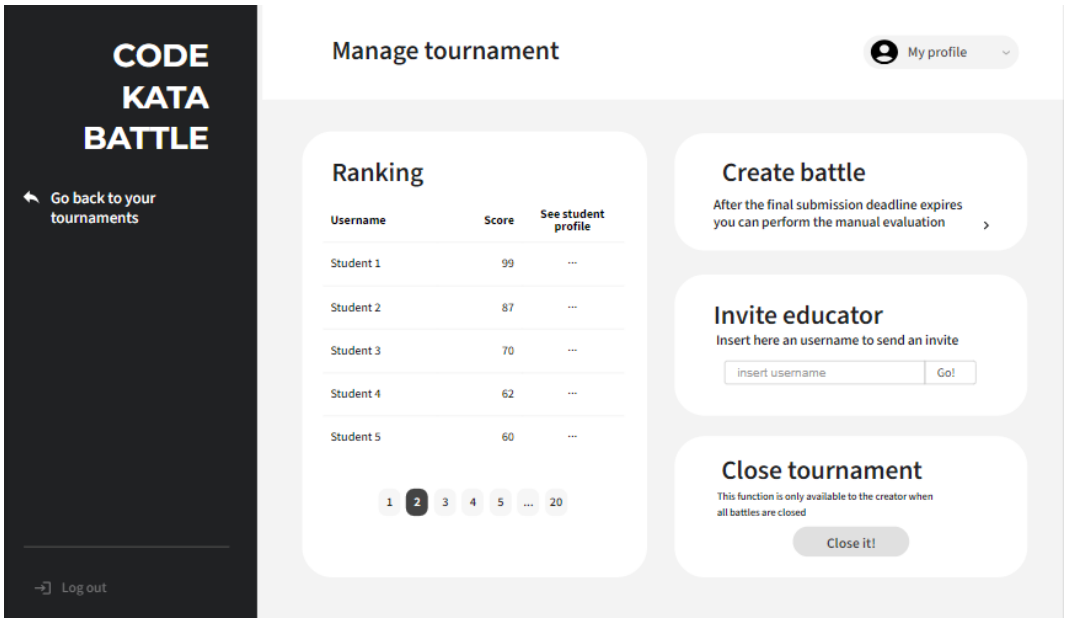

Figure 3.10: Manage tournament

By compile all the fields in the "Create a battle" page in figure 3.11 the educator is able to create a battle in the tournament he/she has selected before.

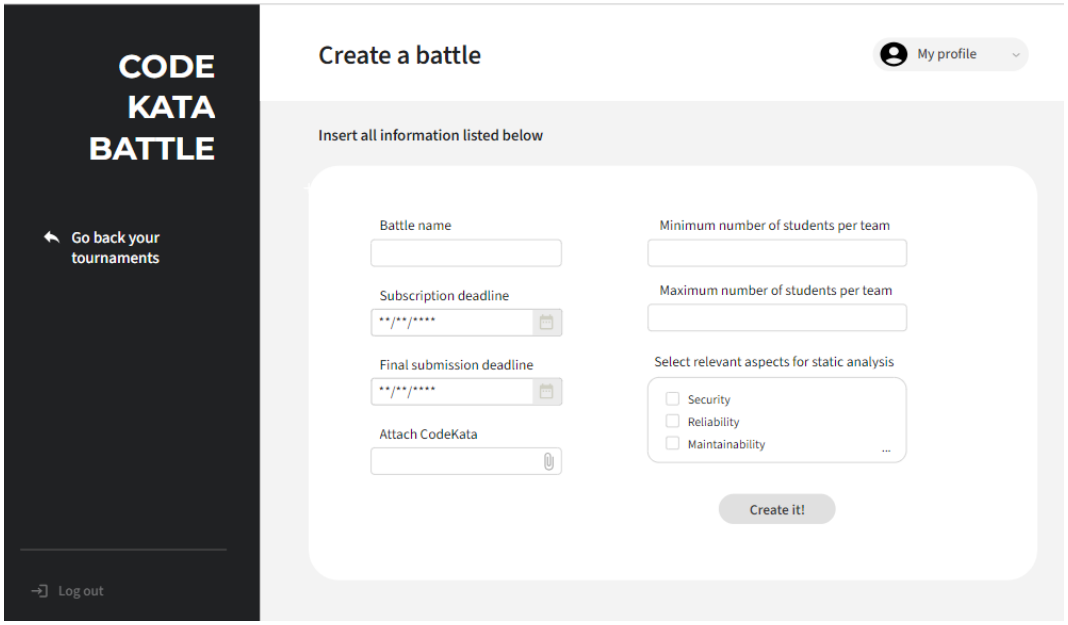

Figure 3.11: Create battle

From the "My battle" page in figure 3.12 the educator is able to manage them. By clicking on one battle the educator will enter the page "manage battle" in figure 3.13.

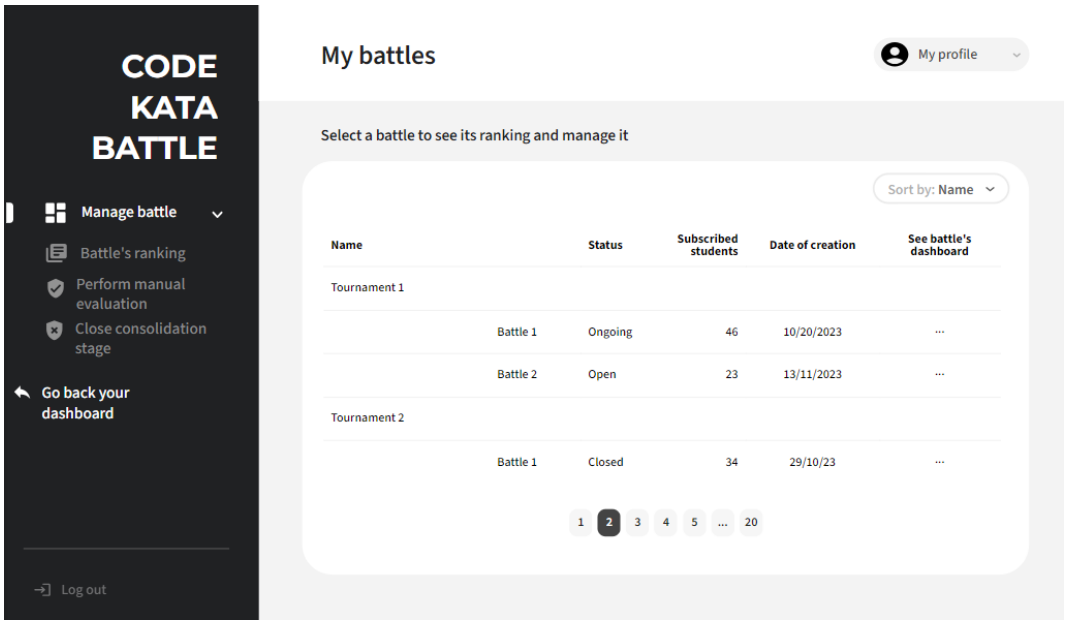

Figure 3.12: Educator's battle

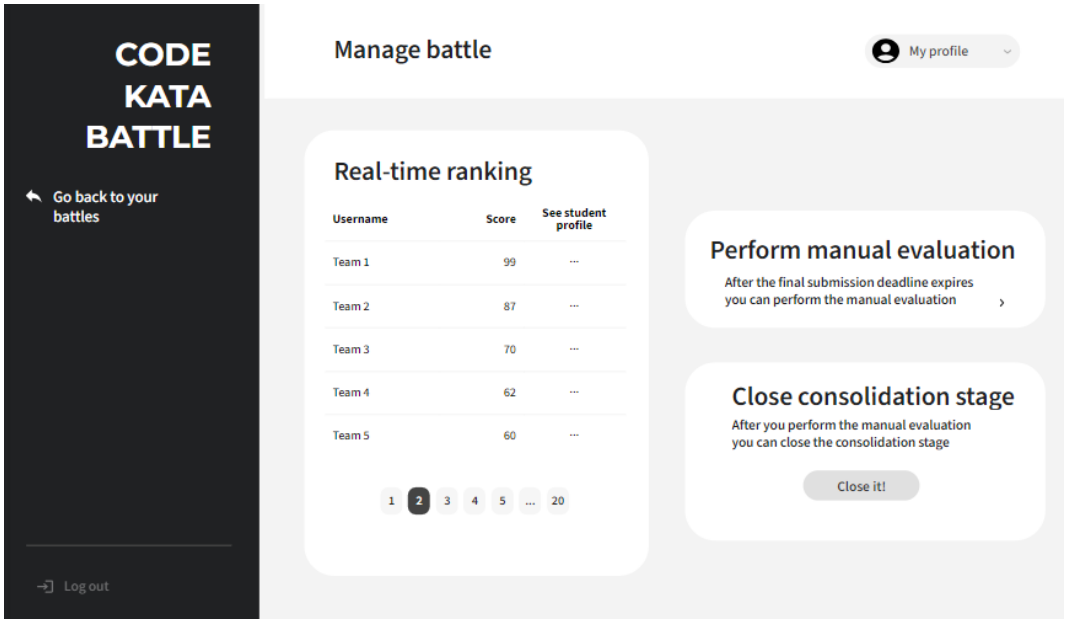

Figure 3.13: Manage battle

The educator can perform the manual evaluation only for his/hers battles. As in figure 3.14 the educator is requested to insert an evaluation for each student enrolled in the battle.

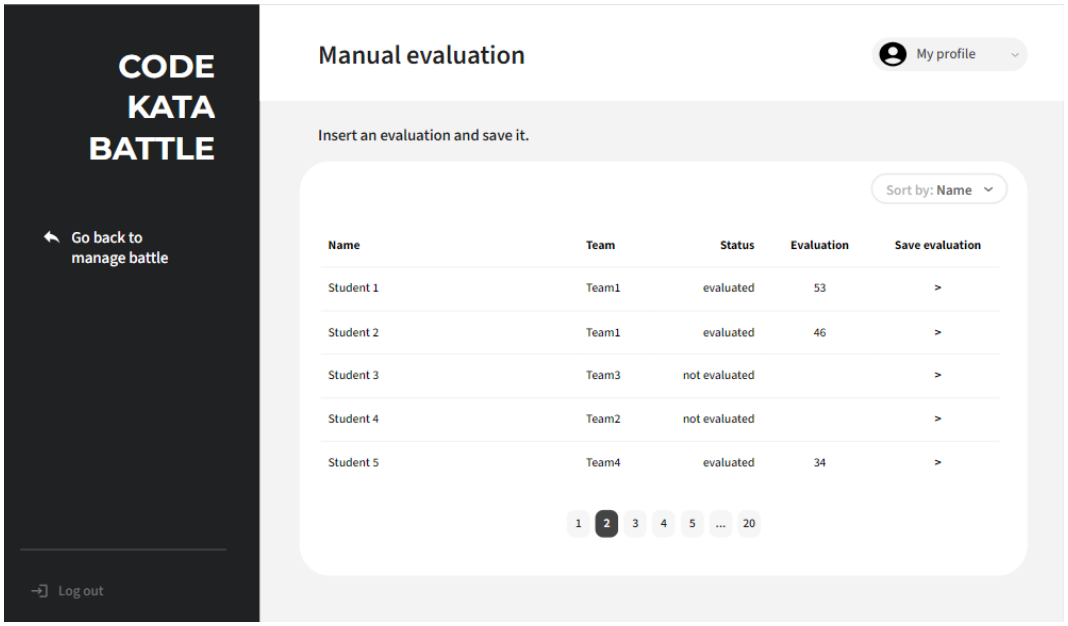

Figure 3.14: Manual evaluation

Used by the students.

From the general dashboard the student will be able to use the CKB platform in all its functionality.

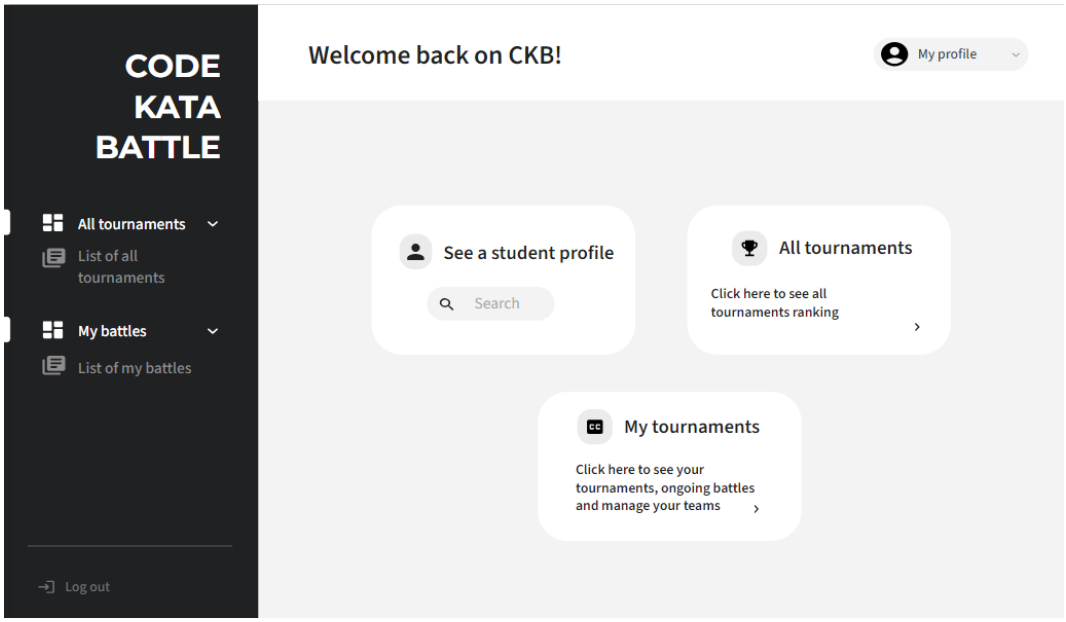

Figure 3.15: Student's dashboard

From the "My tournaments" page in figure 3.16 the student is able to manage his/hers battles by clicking on one of them. The student will enter the page "manage battle" in figure 3.17 from which he/she can manage his tournaments

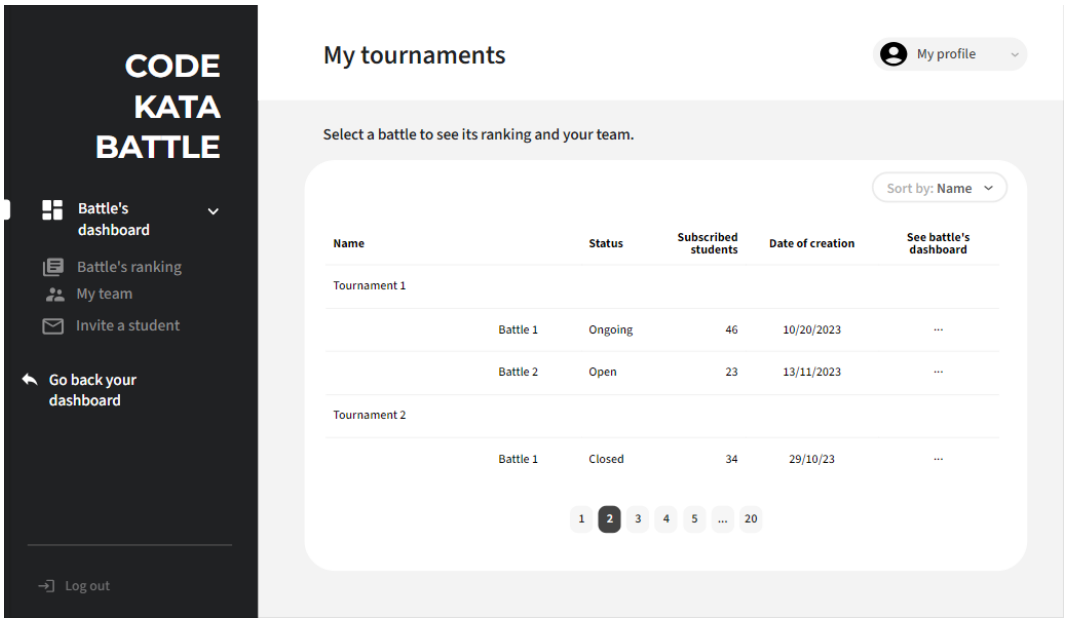

Figure 3.16: Student's tournaments

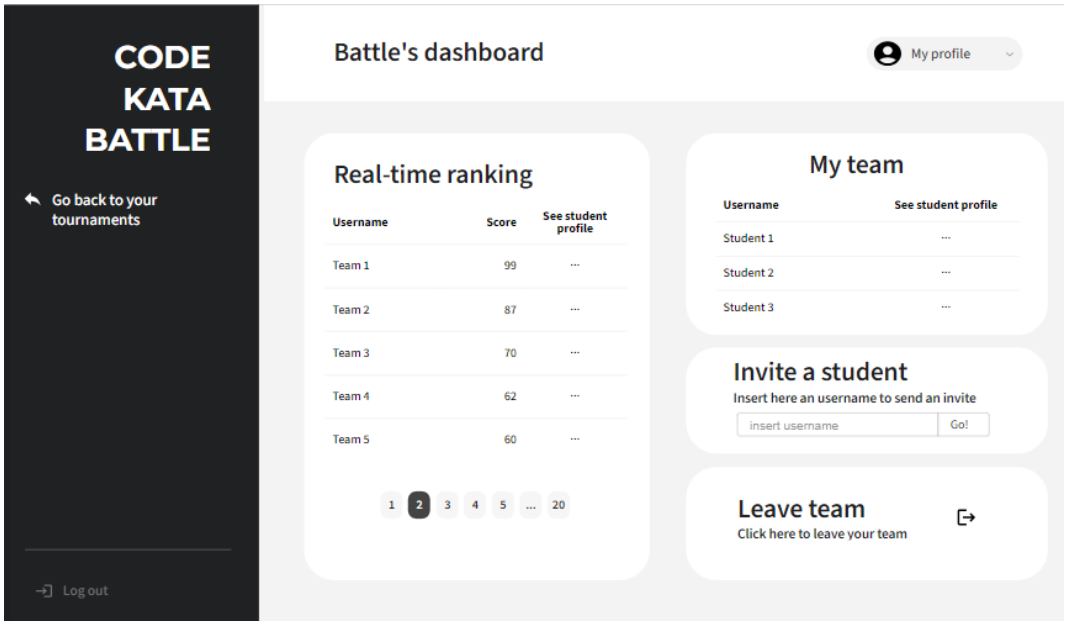

Figure 3.17: Battle dashboard

Chapter 4

Requirements traceability

This section explains which components are needed to meet each requirement stated in the RASD.

Users

R.1 The system shall allow users to register on the platform with their personal information.
Web platform, Dashboard manager, Users access manager

R.2 The system shall allow registered users to log in the platform with valid credentials.
Web platform, Dashboard manager, Users access manager

R.3 The system shall allow all users to see the list of ongoing tournaments.
Web platform, Dashboard manager, Tournament manager

R.4 The system shall automatically evaluate submissions.
Automatic evaluation manager

R.5 The system shall allow users to be able to monitor ranking in real-time during battles and tournaments.
Web platform, Dashboard manager, Tournament manager, Battle manager, Automatic evaluation manager

R.6 The system shall allow users to see other students' profile.
Web platform, Dashboard manager, User access manager

Educators

R.7 The system shall allow the educators to create tournaments.
Web platform, Dashboard manager, Tournament manager

R.8 The system shall allow the educators to manage their tournaments, in particular invites other collaborators and ends the tournament.
Web platform, Dashboard manager, Tournament manager

R.9 The system shall allow the educators to create code kata battles within a tournament.
Web platform, Dashboard manager, Battle manager

R.10 The system shall allow educators to manually evaluate students (if needed) right after a battle closes, during the consolidation stage.
Web platform, Dashboard manager, Battle manager

R.11 The system shall allow educators to create badges.
Web platform, Dashboard manager, Tournament manager

R.12 The system shall allow educators to define rules and variables for a badge.
Web platform, Dashboard manager, Tournament manager

Students

R.13 The system shall allow students to subscribe in tournaments.
Web platform, Link manager

R.14 The system shall allow students to enroll in a battle within a tournament.
Web platform, Link manager

R.15 The system shall allow students to form team for battles, by inviting other students.
Web platform, Dashboard manager, Battle manager, Link manager

R.16 The system shall notify the students about new tournaments within 10 seconds.
Mail manager

R.17 The system shall notify the students about upcoming battles in tournaments in which they are subscribed within 10 seconds.
Mail manager

R.18 The system shall notify the students when they receive an invite to participate in a team within 10 seconds.
Mail manager

R.19 The system shall notify the students when the GitHub repository of a battle is available.
Mail manager

R.20 The system shall notify the students when the final battle rank became available within 10 seconds.
Mail manager, Automatic evaluation manager

R.21 The system shall notify the students when the final tournament rank became available within 10 seconds.
Mail manager

Chapter 5

Implementation, Integration and Test Plan

5.1 Implementation plan

The CKB system will be implemented following a bottom-up approach, taking into account the dependencies between the components within the same subsystem. In this way an incremental integration is promoted, making bug tracking easier because it allows testing of the intermediate outcomes. This strategy allows different teams to work in parallel on implementing different features.

5.1.1 Component integration and testing

This section explains which components are implemented at each stage, as well as how they are integrated and tested.
Since some components may not work in isolation, driver and stub components will be realized to simulate the behaviour of the missing modules.

In the first stage (in figure 5.1) the Model and the Entity Manager are implemented and unit tested, using a driver for the components that are missing. These components handles all the major functionalities of the system and all the interactions with the database.

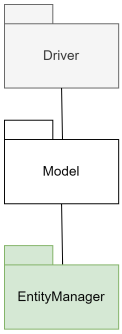

Figure 5.1: Stage 1

In the second stage (in figure 5.2) the components responsible for the sign up and log in of users are implemented, since some components will need this features in the next stage. The Email Manager component is still not implemented and a stub is used to simulate it.

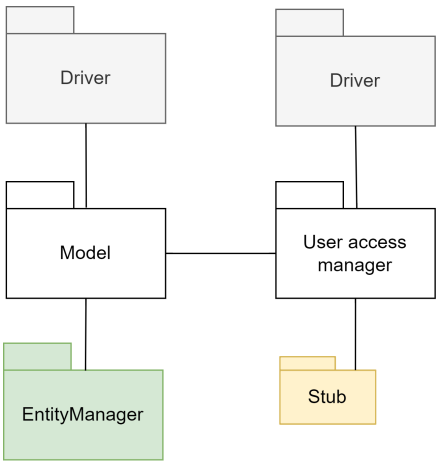

Figure 5.2: Stage 2

In the third stage (in figure 5.3) the components responsible for the creation and management of tournaments and battles are implemented. A driver is used to simulate the interfaces in order to test the new components.

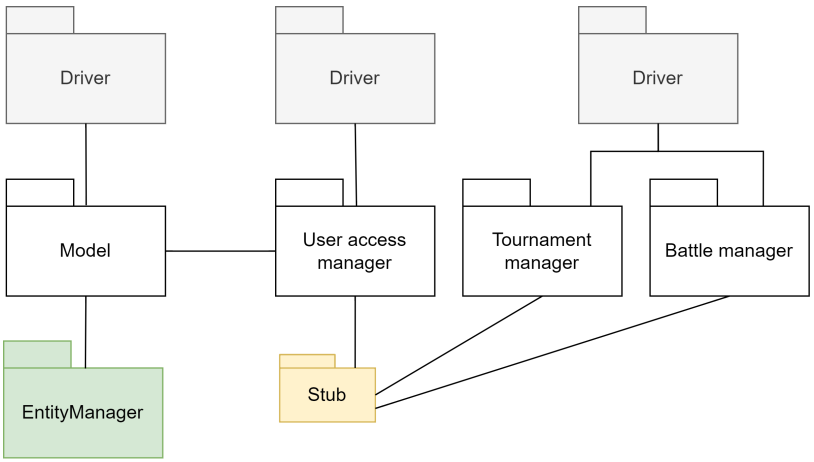

Figure 5.3: Stage 3

In the fourth stage (in figure 5.4) the Automatic Evaluation manager is implemented and another stub is used to simulate the external components (GitHub and static analysis tool API) in order to unit test it.

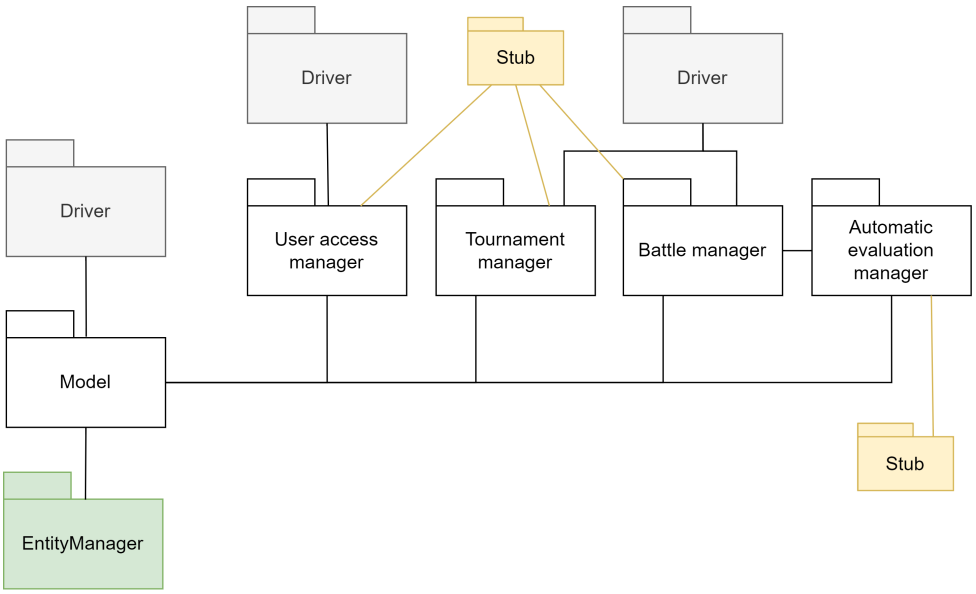

Figure 5.4: Stage 4

In the fifth stage (in figure 5.5) the Mail Manager component is implemented and unit tested. In this way the entire CKB system is working, only the user interfaces are missing.

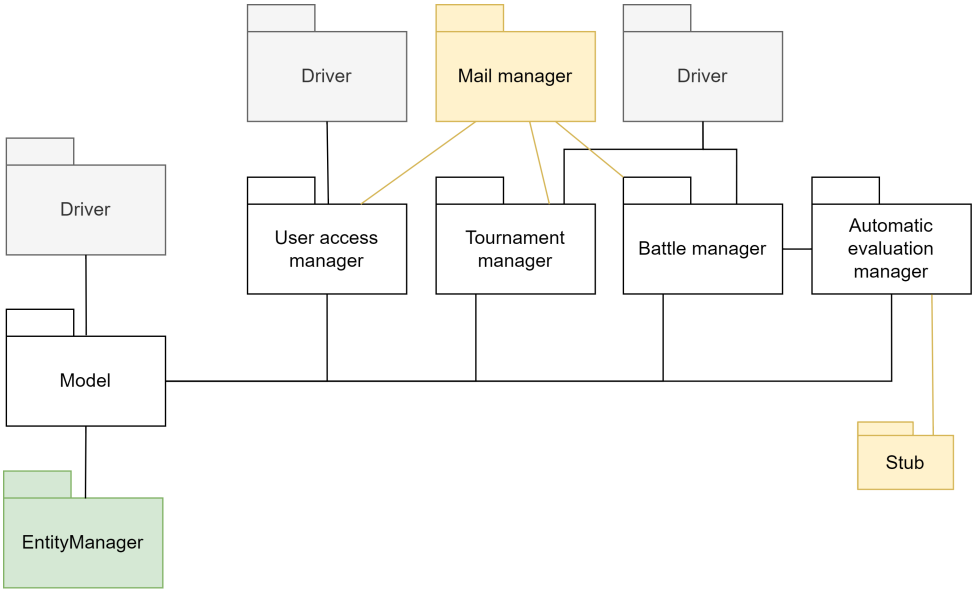

Figure 5.5: Stage 5

In the sixth stage (in figure 5.6), now that all CKB system is working properly,

the Dashboard Manager is implemented and tested, as well as the CKB web platform. The whole system has been implemented.

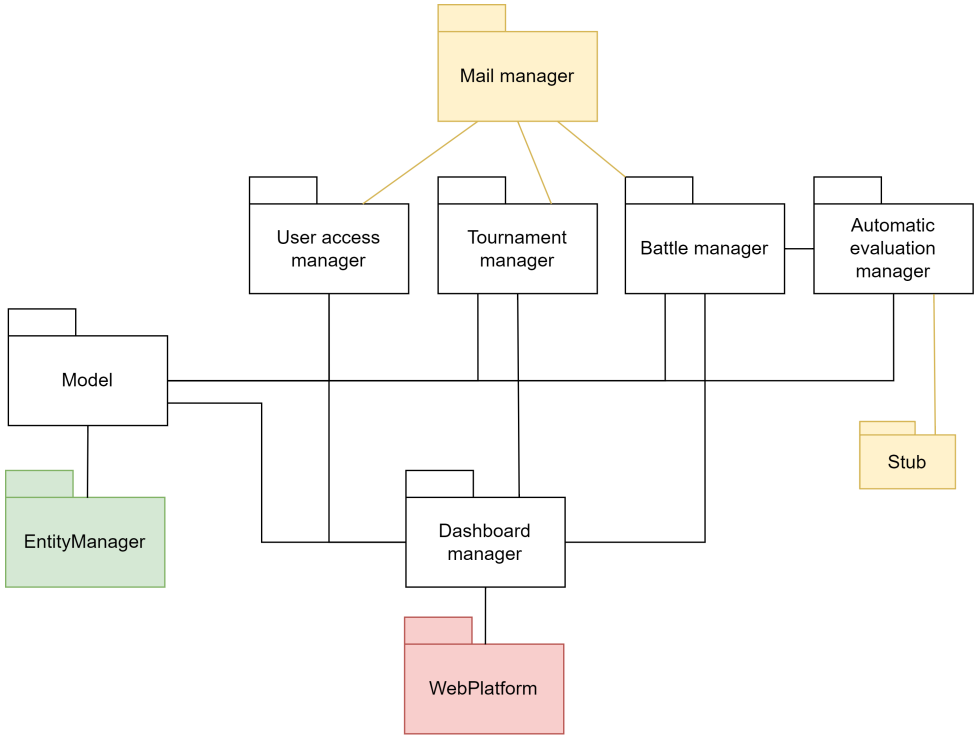

Figure 5.6: Stage 6

5.2 System testing

Once the CKB system has been implemented and unit tested, it is important that the system will be tested as a whole in order to verify that all features works well. Since it is considered that external services are reliable, they do not need to be tested.

Initially, an alpha test of the CKB system will be conducted by the developers in their production environment. A large number of test cases covering at least 80% of the code will be used to verify all functionalities. The system testing is composed of the following testing steps:

- **Integration testing**: to ensure that the integration of the CKB system with the external services works correctly.
- **Functional testing**: to verify that the system meets the functional requirements stated in the RASD.
- **Acceptance testing**: to verify that the system meets the users requirements. It simulates the entire user flow, from both the educator and student view.
- **Security testing**: to identify potential security vulnerabilities.

- **Performance testing**: to evaluate the system performance and response under load. The purpose is to spot any inefficiencies that may affect the system's performance.
- **Usability testing**: to ensure that the user interface is intuitive and user-friendly (regarding the web platform).
- **Stress testing**: to verify that the system can recover gracefully after a failure. It simulates data loss and the restoration of the system as well as stressful or high-load conditions.

At this stage the CKB system should be as close as as possible to the final product. Subsequently a beta test will be executed with the support of a group of voluntary educators and students that will try the web platform independently, and send reports to the developers. This is important in order to verify the correct behavior of the system in a real environment.

5.3 Additional specifications on testing

Every time a new feature is added, the CKB system needs to be tested to check the presence of bugs. In addition, the system testing described in section 5.2 has to be performed to confirm that the system as a whole is correct.
During the development of the system it is important to receive regular feedback from stakeholders and users. In this way the correctness of the system is guaranteed along the creation.

Chapter 6

Effort Spent

6.1 Teamwork

Task	Hours
Initial briefing	2

6.2 Eliahu Itamar Cohen

Task	Hours
Document Setup	1.5
Chapter 1: Introduction	0
Chapter 2: Architectural design	12
Chapter 3: User interface design	1
Chapter 4: Requirements traceability	0
Chapter 5: Implementation, integration and test plan	0
Total	**14.5**

6.3 Marco Gervatini

Task	Hours
Chapter 1: Introduction	0
Chapter 2: Architectural design	6.5
Chapter 3: User interface design	0
Chapter 4: Requirements traceability	0
Chapter 5: Implementation, integration and test plan	0
Extra activities	2
Total	**8.5**

6.4 Caterina Motti

Task	Hours
Document Setup	1
Chapter 1: Introduction	1
Chapter 2: Architectural design	15
Chapter 3: User interface design	8.5
Chapter 4: Requirements traceability	1
Chapter 5: Implementation, integration and test plan	6.5
Extra activities	4
Total	**37**

Chapter 7

References

- All diagrams have been made with https://www.draw.io
- All mockups have been made with https://moqups.com/